CHAPTER 25

The Web-Based Bulletin Board System

OVERVIEW

There are many uses for a bulletin board system on the Web. A BBS can be used to make information continually available that would normally be shared only during infrequent meetings. Within a BBS, users can discuss topics, reading the responses to their ideas at their leisure.

On the Internet, vendors can use a BBS to discuss their products or offer technical support. For example, people can search a BBS for solutions to a problem that may have been brought up by another customer. Additionally, users on the Internet can set up a BBS spontaneously to discuss things that interest them or just for fun. For all these reasons and more, a Web-based bulletin board system (WebBBS) is an ideal asset in almost any Web site.

The ability of people to post messages to each other in a public BBS is a natural evolution in the information sharing of the Internet. Since

the beginning of the Internet, electronic mail has allowed people to exchange information by sending private messages. List servers allow people to subscribe to a sort of "mailing list" where everyone who subscribes gets a copy of every E-mail message sent to the list. Unfortunately, because electronic mail was not designed for large-scale discussion groups, list servers have not proven to be the ideal solution for message sharing. In response to these shortcomings, Internet *newsgroups* were formed. They let people post messages that can then be reviewed by other people at their own pace.

However, not everyone on the Internet has the capability to access Internet newsgroups. In addition, the content of newsgroups tends to be replicated and transferred to sites all over the world. The information generated in some types of discussion groups is not appropriate for worldwide dissemination. For example, a technical support discussion about a small vendor's product may be of use only to 100–200 people in the world—hardly a good case for taking up valuable Internet newsfeed space. Also, with the exception of the "ALT" (Alternative) newsgroups, setting up a newsgroup on the Internet takes a great deal of time and effort.

A discussion board that is local to a particular Web site solves these problems. For one thing, because the discussion board is local to your Web site, you have complete control over the information stored in it. This is an important point, especially for vendors who want to avoid unofficial product information getting out on the Internet. Additionally, a local WebBBS does not have to go through a voting process to get approved; you just set it up. Finally, anyone who has a Web browser can view the WebBBS. There is no need to get an account with an Internet service provider that supplies newsgroup access.

The Web-based discussion board, also known as WebBBS, that accompanies this book has a variety of features that compare favorably with many of the core capabilities found in popular Internet newsgroup readers and servers. The main feature of any discussion forum, including WebBBS, is that the BBS allows a user to post messages as well as post replies to existing messages. The BBS keeps track of which messages are posts and which ones are replies and displays them in a hierarchical tree-like fashion. Posts that start new topics are at the top of each tree, and the

replies are shown indented beneath the original posts. Figure 25.1 shows an example of how posts and their replies are displayed on WebBBS.

Figure 25.1 *A sample listing of posts using WebBBS.*

WebBBS is fully integrated with the user authentication library discussed in Chapter 9. If you opt to turn on user authentication, the user's last name, first name, and E-mail address are automatically placed into the message headers when the user logs in to the message forum. In addition, because authentication allows you to keep track of individual users, WebBBS allows users to view only messages they have not read instead of subjecting them to the full list of messages each time they log in.

Even when authentication is turned off, there are still other options to restrict the volume of messages viewed by the user. WebBBS allows a user to restrict the number of messages through a variety of query mechanisms. The user can conduct a keyword search and display only those

messages that satisfy the search. In addition, the search can be specified as a pattern match or an exact match search. The user can also select a range of dates within which the viewed messages were posted. For example, a user could choose to see only messages posted between "6/1/96" and "6/10/96." In addition, the user can select a range of posts based on age in days, and choose to see, for example, only those messages posted between two days ago and four days ago. These powerful query mechanisms allow a great deal of flexibility so that users can view just the information and posts that they are looking for.

From an administrator's point of view, WebBBS stores messages in a simple format. Each message forum is stored in its own directory, with each post corresponding to a single file. Each filename consists of a unique message number, a hyphen, a message number that the message is replying to, and an **.MSG** extension. Message number zero is referred to as a message that does not exist. Thus, the first post on the system that is not replying to any previous post would be **000001-000000.msg**. Message number 5 on a system replying to message number 3 would have a filename of **000005-000003.msg**. Figure 25.2 shows a directory listing that contains the message filenames corresponding to the messages listed in Figure 25.1.

```
% ls -l
total 12
-rw-rw-rw-   1 wwwadmin wwwadmin      77 Jul 22 13:44 000001-000000.msg
-rw-rw-rw-   1 wwwadmin wwwadmin     139 Jul 22 13:44 000002-000001.msg
-rw-rw-rw-   1 wwwadmin wwwadmin     213 Jul 22 13:44 000003-000002.msg
-rw-rw-rw-   1 wwwadmin wwwadmin     213 Jul 22 13:44 000004-000002.msg
-rw-rw-rw-   1 wwwadmin wwwadmin     101 Jul 22 13:45 000005-000000.msg
-rw-rw-rw-   1 wwwadmin wwwadmin     199 Jul 22 13:45 000006-000005.msg

% _
```

Figure 25.2 *Listing of filenames corresponding to messages displayed in Figure 25.1.*

The posts are stored in this format to allow WebBBS to look at the messages for a given forum and automatically know which messages were the most recently posted (the ones with the highest message numbers) and which messages replied to which other messages. In this way, WebBBS develops a hierarchical view of the messages. Because this view is generated dynamically, posts can easily be removed without the need to update any other files. Thus, a WebBBS administrator can turn on features such as automatic message deletion when a post reaches a certain age or when the number of posts in a directory exceeds a certain number. In addition, a WebBBS administrator can manually delete a message if it has been posted inappropriately to the WebBBS forum.

Another useful feature of WebBBS is the ability to define forums on the fly. The administrator need only create a new directory for each forum and then add the forum to a list of other forums in the setup file.

If a user is using a Web browser, such as Netscape version 2.0, that is compatible with the new `<INPUT TYPE=FILE>` HTML form tag, the administrator can allow attachments to be uploaded along with the contents of a post. This feature can be useful, because it allows people to transfer files to one another. For example, product updates and patches can be posted directly on a technical support BBS and be available for users to download.

Installation and Usage

The WebBBS files on the accompanying CD-ROM install into a directory called **WebBBS**. The files and subdirectories associated with this application along with their required permissions are shown in Figure 25.3.

WebBBS is the root application directory. It must be readable and executable by the Web server. In addition to the application files, the **Msg_open**, **Library**, **Msg_CGI**, **Attach**, **Images**, **Users**, and **Sessions** subdirectories are located here.

bbs_forum.cgi is the main CGI script that performs all the BBS functions, including reading and posting new messages. This file must be readable and executable.

bbs.setup is a setup file for the BBS. This file must be readable.

bbs_html_error.pl contains Perl code that outputs an error message to the user if anything goes wrong. This file must be readable. Figure 25.4 shows an example of this page on a Web browser.

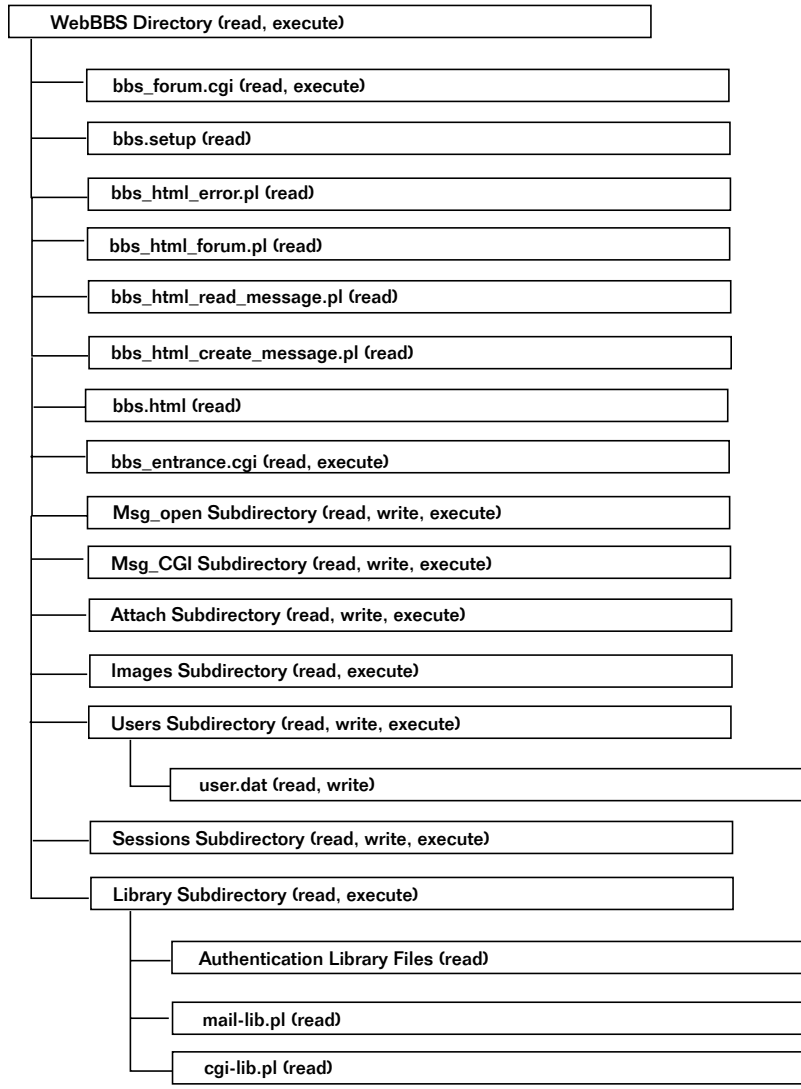

Figure 25.3 *BBS script directory structure and permissions.*

Figure 25.4 *bbs_html_error.pl output displayed on a Web browser.*

bbs_html_forum.pl contains Perl code that outputs the list of messages in a particular forum. This file must be readable. Figure 25.1 shows an example of the output from this file.

bbs_html_read_message.pl contains Perl code that outputs an HTML version of a message that has been previously posted on the BBS. This file must be readable. Figure 25.5 shows an example of the HTML output when a message is read.

bbs_html_create_message.pl contains Perl code that outputs an HTML form for creating a message on the BBS. This file must be readable. Figure 25.6 shows the output of the HTML for posting a message to a BBS.

Figure 25.5 *bbs_html_read_message.pl output displayed on a Web browser.*

bbs.html is a sample HTML file showing how to access a forum from an HTML file. This file should be placed in a directory where your Web server is allowed to read HTML files. In addition, the file should be readable. It is discussed further in the "Running the Script" section.

bbs_entrance.cgi is a sample script that outputs an HTML form where the user chooses various options. These options are then posted to **bbs_forum.cgi** when the user presses a submit button. This file should be readable and executable. In the "Running the Script" section, **bbs_entrance.cgi** is discussed further.

The **Library** subdirectory stores all the external libraries the script must access. This directory should be readable and executable, and all the files in it should be readable. These files include all the files associated with

the authentication library discussed in Chapter 9 (for logging a user on to the BBS), **cgi-lib.pl** (for processing HTML forms), and **mail-lib.pl** (for sending E-mail).

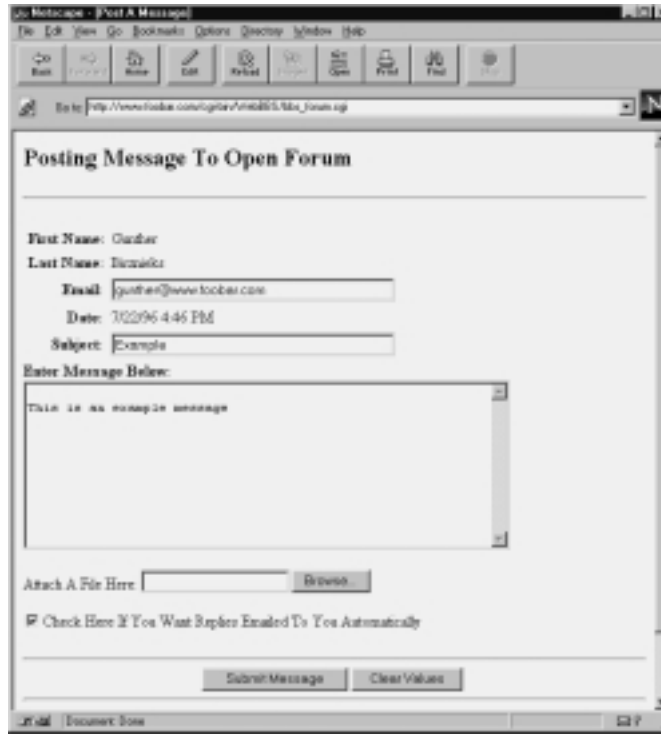

Figure 25.6 *bbs_html_create_message.pl output displayed on a Web browser.*

The **Attach** subdirectory is used to store the attachments that are uploaded by users of the WebBBS. If your Web server is set up in a way that disallows nonexecutable files in CGI directories, you may need to place this directory where the user's Web browser can download files directly. In addition, you may want to place this directory in a non-CGI area so that no one will be able to upload a file to a CGI directory and then run the program using a Web browser. This directory should be readable and executable by the Web server.

The **Images** subdirectory holds the images of the buttons used by the WebBBS. You may need to place this directory where HTML files can read images if image files are not allowed to be mixed with CGI executables on your Web server. This directory must be readable and executable. The files within it must be readable.

The **Users** subdirectory is used by the authentication library to store the user list. This directory must be readable, writable, and executable. A **user.dat** file is created in this directory by the script the first time a user registers for the Web site.

Although we discuss the use of the authentication library with regard to the BBS, the default distribution on the accompanying CD-ROM comes with authentication turned off in the bbs.setup file. The ramifications of using authentication will be discussed later.

The **Sessions** subdirectory is used by the authentication library to store the files related to each user's session after the user first logs on to the site. This directory must be readable, writable, and executable.

The **Msg_open** subdirectory is used by **bbs_forum.cgi** to store messages for the Open Forum message board. This is one of the default message forums set up in **bbs.setup**. This directory must be readable, writable, and executable.

Directories that contain WebBBS messages will also contain [username].dat files if authentication is enabled in the setup file. These files keep track of each user's last read message number.

The **Msg_CGI** subdirectory is used by **bbs_forum.cgi** to store messages for the CGI talk message board, just as **Msg_open** does for the Open Forum. This directory must be readable, writable, and executable.

Server-Specific Setup and Options

The **bbs.setup** file contains the configuration variables for **bbs_forum.cgi**. The following is a list of these setup items.

`@forums` is a list of forum names. These names are descriptive names of the forums that are available on the BBS. For example, if you had one forum for discussing open topics and another for discussing CGI programming, this variable would be set to `("Open Forum", "CGI Programming Forum")`.

`@forum_directories` is a list of the directory paths that correspond to the list of forums in `@forums`. Each of these directories stores the messages related to its own forum. An example list of values for `@forum_directories` that matches the preceding example would be `("Msg_open", "Msg_CGI")`.

`@forum_variable` is a list of forum variable names related to each BBS forum. Whenever **bbs_forum.cgi** is called, it must have the variable `forum` sent to it. This variable is equal to the name in the `@forum_variable` array. Each element of the array corresponds to the forums listed in the `@forums` array. Because the values here are variable names, you should use lowercase text, underscores instead of spaces, and no special characters. For example, `@forum_variables` could be set to `("open", "cgi")`, which corresponds to the `"Open Forum"` and `"CGI Programming Forum"` values of `@forums`.

`$bbs_script` is the **bbs_forum.cgi** script name. Most systems keep this variable set to `bbs_forum.cgi`, but in some systems the script must be renamed. For example, some Windows NT Web servers require that the script name have a **.bat** extension. The `$bbs_script` variable is used by **bbs_forum.cgi** to make references to itself from URLs it generates.

`$allow_user_attachments` is set to `on` if you are allowing people to upload to the BBS.

`$maximum_attachment_size` is the size in bytes of the maximum amount of data you want sent to the WebBBS script from the user's Web browser when a file is uploaded.

`$attach_dir` is the path to the directory where attachments are uploaded. This should be set to a directory that allows users to download the files directly with their Web browser. On some Web servers, directories that can run CGI scripts cannot view HTML files. Generally, the same rules apply to the downloading of files.

`$attach_url` is the URL that is used to prefix attached files. In the event that the attached files are uploaded to a directory where HTML files can be read as opposed to being able to execute CGI scripts, this URL must point a user's Web browser to the new directory.

`$display_only_new_messages` is `on` if you want to keep track of the last new message that each user has read.

There are several requirements for this to take place. First, you must use authentication to record the user name when the user enters the forum. Second, the form variable `use_last_read=on` must be in the first URL call to bbs_forum.cgi. The user may want to see all the new messages, so leaving off the `use_last_read` form variable allows the user to see everything even if the last read message has been recorded.

`$display_thread_depth` is the number of descending nodes to be displayed in the message listing. For example, with a thread depth of 2, the user sees only the original posts and the replies indented beneath the message. With a thread depth of 3, the replies to the replies are seen, with further indentations. If you recall, Figure 25.1 shows an example of several message threads that are indented with respect to one another.

`$prune_how_many_days` is the number of days after which a message is considered too old to leave on the system. These messages are deleted. If this variable is set to zero, messages are not removed on the basis of age.

`$prune_how_many_sequences` is the maximum number of messages you want to leave on the system before the oldest ones are deleted. For example, if you specify this number to be 10, then only 10 messages will be allowed per forum. In this case, after the 11th message is posted, message number 1 would be deleted. Setting this value to zero means that you do not want any messages deleted on the basis of a maximum number of messages to keep on the system.

`$bbs_buttons` is the URL prefix for the directory that contains the BBS button images. By default, these images are located in the **Images** subdirectory underneath the directory in which the BBS script is installed. You may need to move these images to another directory if only CGI- executables are allowed inside this directory.

`$no_html` is set to `on` if you want to filter HTML from a user's post. It is a good idea to prevent users from posting HTML within their messages,

because it can do nasty things. For example, users might leave off a closure tag in the subject, such as </H1> if they are including a header. This omission would make all the other messages inherit the header attribute.

$no_html_images is set to on if you want to prevent people from referencing images in their messages. Setting $no_html to on also filters out image-related HTML tags, so setting this variable, $no_html_images, to on is for administrators who want to continue to allow HTML posting but not images. In other words, $no_html being set to on includes filtering out all image tags, so $no_html_images need not be set if $no_html is already on.

$use_list_element is set to on if you wish the message threads to be displayed with HTML-style bullets. Displaying with bullets looks nicer but takes up more space on the screen.

$allow_reply_email is set to on if you want to allow users to choose whether to get E-mail notifications of replies to their posts.

$force_reply_email is on if you want to force users to get E-mail notifications of replies to their posts. In this case, users are not given a choice. They will always get the notifications.

$send_reply_email needs to be on for either of the preceding two variables to work. If $send_reply_email is off, then E-mail will not be sent even if the message specifies that an E-mail reply should be sent. This variable allows the administrator to turn off the E-mail reply feature even after people have posted messages asking for E-mail–based replies.

If authentication is being used, the user's first name, last name, and E-mail are specified at login. If this is the case, setting $force_first_name, $force_last_name, and $force_email to on forces those fields be to remain the same whenever a user posts on the BBS. Normally, a user can enter his or her name and E-mail address information when posting a message, but setting these variables on disallows that.

$require_subject , $require_first_name, $require_last_name, and $require_email are set to on if you want to make sure that the user enters something into these fields instead of leaving them blank. In other words, setting these variables to on makes them required fields; a post will not be created without them.

If replies are to be E-mailed, then `$from_email` must be set to an E-mail address from which you want the replies generated. Usually, you set this variable to the E-mail address of the Web server or your own E-mail address on the system.

The remaining variables are authentication library settings. By default, `$auth_cgi` and `$auth_server` are set to `off`. However, enabling one of these types of authentication enables the user's first name, last name, and E-mail information to automatically be entered when the user posts a message. Authentication also enables the BBS to track the last read message through the acquired username. Chapter 9 discusses the various authentication variables in detail.

The following is an example of all the setup variables in the **bbs.setup** file.

```
@forums = ("Open Forum", "CGI Programming");
@forum_directories = ("Msg_Open", "Msg_CGI");
@forum_variable = ("open", "cgi");

$bbs_script = "bbs_forum.cgi";
$allow_user_attachments = "on";
$maximum_attachment_size = 500000;
$attach_dir = "Attach";
$attach_url = "Attach";

$display_only_new_messages =      "on";
$display_thread_depth =      8;
$prune_how_many_days =           5;
$prune_how_many_sequences =      100;
$bbs_buttons = "Images";

$no_html = "off";
$no_html_images = "on";

$use_list_element = "on";
$allow_reply_email = "on";
$force_reply_email = "off";
$send_reply_email = "on";

$force_first_name = "on";
$force_last_name = "on";
$force_email = "off";
```

```
$require_subject = "off";
$require_first_name = "on";
$require_last_name = "on";
$require_email = "on";

$from_email = "gunther@foobar.com";

$auth_lib = ".";
$auth_server =                  "off";
$auth_cgi =                     "on";
$auth_user_file =               "Users/user.dat";
$auth_alt_user_file =           "";
$auth_allow_register =          "on";
$auth_allow_search =            "on";
$auth_default_group =           "normal";
$auth_generate_password =       "off";
$auth_check_duplicates =        "on";
$auth_add_register =            "on";
$auth_email_register =          "off";
$auth_admin_from_address = "wwwadmin\@fooobar.com";
$auth_admin_email_address = "gunther\@foobar.com";
$auth_session_length = 2;
$auth_session_dir = "./Sessions";
$auth_register_message =
        "Thanks, you may now logon with your new
        username and password.";
$auth_password_message =
        "Thanks for applying to our site,
         your password is";
@auth_extra_fields = ("auth_first_name",
                      "auth_last_name",
                      "auth_email");
@auth_extra_desc = ("First Name",
                    "Last Name",
                    "Email");
$auth_logon_title = "Submit BBS Logon";
$auth_logon_header = "Enter BBS Logon Information";
```

Using Other Setup Files

The BBS script has the ability to reference another setup file in case the default **bbs.setup** file does not meet the needs of every forum. For example, although **bbs.setup** accommodates multiple forums, you may want to assign a different automatic removal of messages policy for each one. On the Open Forum, you may want to delete messages older than 10 days, but

on a CGI Programming forum, you may not want to delete any messages.

You can do this by using another setup file that is loaded with the same variables defined by **bbs.setup**. **bbs.setup** is always read into the **bbs_forum.cgi** script. However, if you set the `setup` variable on the URL link to the script as a form variable, **bbs_forum.cgi** will read a setup file on the basis of that variable. For example, if you specified the call to **bbs_forum.cgi** as

```
http://www.foobar.com/cgi-
bin/WebBBS/bbs_forum.cgi?forum=open&setup=test
```

then the **test.setup** file would be loaded by the BBS after the **bbs.setup** file is loaded.

bbs.setup is always necessary. This means that if you choose to override bbs.setup using the `setup` form variable, you need only specify the variables you want changed in the new setup file instead of all the variables originally residing in bbs.setup.

The only variables you cannot override with the second setup file are the file upload variables. To read the second setup file, **cgi-lib.pl** must read the form variables first. Unfortunately, the files are uploaded inside the `ReadParse` routine in **cgi-lib.pl**. Thus, the files are already uploaded by the time the `setup` form variable has been read by `ReadParse`.

Modifying the HTML

The HTML scripts for the core WebBBS script are stored in files that are prefixed with **bbs_html**. These Perl scripts output the HTML forms for posting and reading messages as well as other BBS operations. To modify the cosmetics of the BBS, you need only edit these files. They are discussed in more detail in the "Design Discussion" section.

Running the Script

To use the **bbs_forum.cgi** program, you refer to it along with urlencoded information. The only mandatory URL variable is `forum`. However, several others can be set up. Here is a sample URL for this script if it is installed in the **WebBBS** subdirectory underneath the **cgi-bin** directory:

```
http://www.foobar.com/cgi-bin/WebBBS/bbs_forum.cgi?forum=open
```

A sample HTML file that also has this link is shown next. The link also has a setup variable defined. Figure 25.7 shows what **bbs.html** looks like on a Web browser.

```
<HTML>
<HEAD>
<TITLE>BBS Forum List</TITLE>
</HEAD>
<BODY>
<H1>BBS Forum List</H1>
<HR>
<A HREF=
"bbs_forum.cgi?forum=open&setup=test ">
Enter The Open Forum</A>
<HR>
</BODY>
</HTML>
```

Figure 25.7 *bbs.html displayed in a Web browser.*

Optional URL Variables

There are several optional form variables that **bbs_forum.cgi** recognizes. These include `setup`, `use_last_read`, `keywords`, `exact_match`, `first_date`, `last_date`, `first_days_old`, and `last_days_old`.

`setup` is the form variable for specifying a new setup file for the BBS. It is set equal to the name of the setup file you wish to read minus the **.setup** extension. Thus, if you wish to use **test.setup** to override **bbs.setup**, you would set this variable equal to `test`. Here is an example URL:

```
http://www.foobar.com/cgi-
bin/WebBBS/bbs_forum.cgi?forum=open&setup=test
```

`use_last_read` must be set to `on` if you wish users to see only messages that they have not already seen. For this to work, the BBS must be configured to track last read messages for users, and authentication must be turned on. The following URL turns this variable on:

```
http://www.foobar.com/cgi-
bin/WebBBS/bbs_forum.cgi?forum=open&use_last_read=on
```

`keywords` is a list of keywords that you want the messages to contain. For example, if you set `keywords=hockey`, only messages containing the string "hockey" will be displayed. In addition, the `exact_match` variable can be set to `on` if you want the keyword search to match whole words only. Here is an example URL for searching for "hockey" with an exact match search:

```
http://www.foobar.com/cgi-bin/WebBBS/bbs_forum.cgi?forum=open&key-
words=hockey&exact_match=on
```

`first_date` and `last_date` are the range of dates within which a message can be viewed. For example, if `first_date` is set to `1/1/95` and `last_date` is set to `1/15/95`, only messages between those dates will be seen by the user. A URL for doing this is shown next. Note that `%2F` replaces the forward slash (`/`) symbol, because Web browsers recognize forward slashes only as separations for subdirectories; the forward slash must be translated into its hexadecimal ASCII-code equivalent (`%2F`).

```
http://www.foobar.com/cgi-
bin/WebBBS/bbs_forum.cgi?forum=open&first_date=1%2F1%2F95&last_date=1%
2F15%2F95
```

`first_days_old` and `last_days_old` operate similarly to the date-related variables. This is also a date range on which to view posts, but it is based on the age of a post in days. If `first_days_old` is set to `30` and `last_days_old` is set to `15`, messages between the ages of 30 and 15 days old will be seen by the user. The following example URL illustrates this:

```
http://www.foobar.com/cgi-
bin/WebBBS/bbs_forum.cgi?forum=open&first_days_old=30&last_days_old=15
```

The **bbs_entrance.cgi** program shown next prints a form allowing users to see and set the different options for restricting the view of their messages. Figure 25.8 shows an example of this form.

```
#!/usr/local/bin/perl
print "Content-type: text/html\n\n";
print <<__END_OF_HTML__;
<HTML>
<HEAD>
<TITLE>BBS Version 4.0 Sample Entrance</TITLE>
</HEAD>
<BODY BGCOLOR = "FFFFFF" TEXT = "000000">
<CENTER>
<H1>BBS Sample Entrance</H1>
<HR>
</CENTER>
<FORM ACTION="bbs_forum.cgi" METHOD=POST>
<TABLE BORDER = "1">
<TR>
<TH ALIGN = "left">Forum</TH>
<TD><SELECT NAME = "forum">
<OPTION VALUE = "open">Open Forum
<OPTION VALUE = "cgi">CGI Programming
</SELECT>
</TD>
</TR>
<TR>
<TH ALIGN = "left">
Display messages with what keywords?
</TH>
```

```
<TD>
<INPUT TYPE = "text" NAME = "keywords">
</TD>
</TR>
<TR>
<TH ALIGN = "left">
Exact Match for keyword search?
</TH>
<TD>
<INPUT TYPE = "checkbox" NAME = "exact_match">
</TD>
</TR>
<TR>
<TH ALIGN = "left">
Range of Dates Posted (First date in range to view
messages, e.g., 12/03/98)
</TH>
<TD>
<INPUT TYPE = "text" NAME = "first_date">
</TD>
</TR>
<TR>
<TH ALIGN = "left">
Range of Dates Posted (Last date in range to view
messages, e.g., 12/03/98)
</TH>
<TD>
<INPUT TYPE = "text" NAME = "last_date">
</TD>
</TR>
<TR>
<TH ALIGN = "left">
Range of Age of posts (earliest number of days old to
view msgs)
</TH>
<TD>
<INPUT TYPE = "text" NAME = "first_days_old">
</TD>
</TR>
<TR>
<TH ALIGN = "left">
```

```
Range of Age of posts (latest number of days old to
view msgs)
</TH>
<TD>
<INPUT TYPE = "text" NAME = "last_days_old">
</TD>
</TR>
</TABLE>
<P>
<CENTER>
<INPUT TYPE = "submit"
VALUE = "Enter the BBS with these parameters">
</CENTER>
<BLOCKQUOTE>
<STRONG>Instructions:</STRONG> All the fields that
appear above are optional except for the forum field.
The forum is needed in order to determine which
messages to view. Normally all messages in a forum
are displayed. However, entering values into the above
fields will narrow down the messages that are displayed.
<P>
Entering a keyword will display only messages with that
keyword. The keyword search can also be specified as an
exact match search.
<P>
You can also specify a range of dates to view the
posts. In other words, you can specify to view only
the posts that were created between a certain range
of days.
<P>
In addition to the above date range search, you
can choose to narrow down the age of posts as
a factor of days instead of an actual date range.
For example, if you wanted to view only posts 2 days
old and newer, then you would enter the number 2 into the
"earliest number of days to view messages" field.
</BLOCKQUOTE>
</BODY>
</HTML>
__END_OF_HTML__
```

Figure 25.8 *bbs_entrance.cgi HTML form output.*

BBS Maintenance

After you have been running the WebBBS for a little while, you may find yourself doing occasional maintenance on it. The two most common operations are the deletion of messages and the editing of messages.

Deleting messages is easy. Because the messages are stored in a directory you specify, finding the message to delete is a matter of looking at the directory and removing the file that corresponds to the message.

On UNIX, the rm command removes a file.

Each message is stored as two six-digit numbers separated by a hyphen with an **.msg** extension. If an attachment is associated with the message, it has the same filename except that the extension is **.attach** instead of **.msg**. The first six-digit number is the message number. The second six-digit number is the message number that it is a reply to. If the second six-digit number is 000000, the message is an original post and is not a reply to anything previously posted. Here is an example list of message files:

```
000001-000000.msg
000002-000000.msg
000003-000001.msg
000004-000003.msg
000005-000002.msg
```

The preceding list shows message numbers 1 through 5. The leading zeros are necessary for the BBS programming to sort the messages. (This is discussed later in the "Design Discussion" section). We can see that messages 1 and 2 are original posts, because they have "000000" as the reply-to message number. Message 3 is a reply to message 1, message 4 is a reply to message 3, and message 5 is a reply to message 2.

If you want to delete message 2, simply delete the message. On UNIX, you can use the **rm** command to remove it by typing the following command while you are in the directory containing the messages:

```
rm 000002-000000.msg
```

Remember, if an **.attach** file exists with the same message number, you need to delete it also. Before you delete the **.attach** file, you should look at the contents. It contains a single line that tells you which directory and filename the attachment is stored in so that you can remove the uploaded file as well.

A neat feature of WebBBS is that you do not have to worry about deleting a message that other messages may be replying to. If a message is missing in the hierarchy of replies, WebBBS automatically readjusts the hierarchy in the message list display so that a reply to a post that is deleted becomes a top-level post. For example, if message 3 was deleted, mes-

sage 4 would be seen by the BBS as an original post, because the reply-to message "000003" no longer exists. Thus, the BBS gracefully takes care of reordering the messages if you decide to delete some of them.

To find out the message number of the message you want to delete, you can view the message in the BBS using your Web browser. When you are reading a message, the URL in the location box of your Web browser shows you a `message=` variable that is set to the message number currently being read. Figure 25.5 has a location box that illustrates this.

If you want to edit a message, load the message in any text editor. Be careful to keep the message structure the same. Do not delete a line that contains any header fields, such as first name, last name, E-mail address, date and time of the post, subject, and options. Here is an example of a message in its "raw" form:

```
Gunther
Birznieks
gunther@foobar.com
7/1/96 8:14 AM
test subject
options:email:gunther@foobar.com

This is the message.
```

The first line of the file contains the first name of the poster. The next line contains the last name of the poster. The E-mail address appears on the third line, and the date and time stamp of the message is stored on the fourth line. The subject is on the fifth line of the message.

The sixth line may seem weird because of the `options:` tag. This line is needed because it contains information about the options the user intends for this message. Currently, only one option is implemented: `email`. When the `options:` tag has `email` associated with it, an E-mail will be generated whenever someone replies to the message.

The subsequent lines contain the body of the post. There is nothing unusual about the body of the post. It can contain multiple lines and

continues to the end of the message file. The body of the preceding post consists of a couple of carriage returns for extra spacing plus the text "This is the message."

Design Discussion

WebBBS performs all its functions using the **bbs_forum.cgi** script. These operations include the listing of messages in a forum as well as the creation of those messages. Depending on the value of incoming form variables, **bbs_forum.cgi** determines which procedure to perform. A flowchart of the BBS features is shown in Figure 25.9.

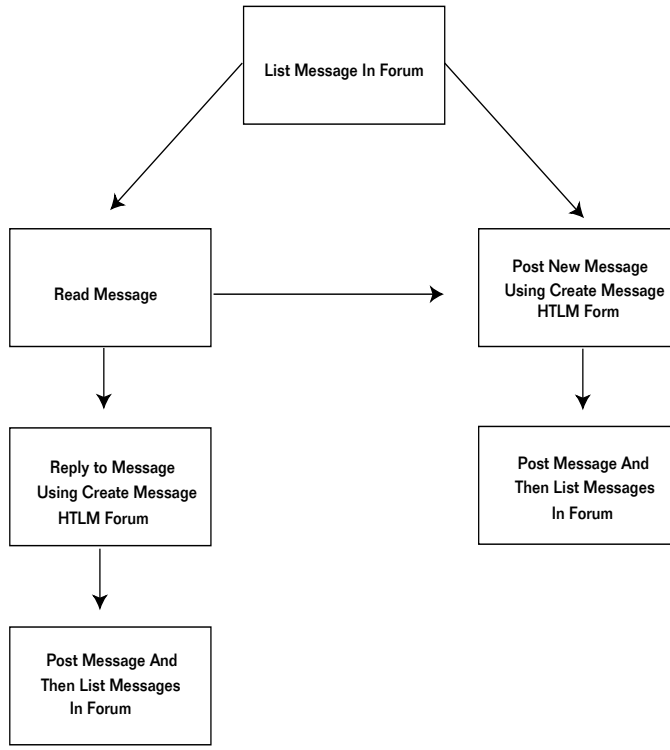

Figure 25.9 *Basic flowchart for the BBS.*

bbs_forum.cgi

The first line of the following code sets up the location of the supporting files of the program. By default, `$lib` is set to the current directory. The core setup variables are loaded from **bbs.setup**. Then **cgi-lib.pl** is loaded.

```
$lib = ".";
require "./bbs.setup";
require "$lib/cgi-lib.pl";
```

The standard "Content-type: text/html\n\n" HTTP header is printed.

```
print &PrintHeader;
```

The next section of code sets up the upload portion of the BBS. If `$allow_user_attachments` has been turned `on` in **bbs.setup**, then uploads are enabled. The `maxdata` variable in **cgi-lib.pl** is set to `$maximum_attachment_size` from **bbs.setup** to restrict the size of a file that is uploaded. The `writefiles` variable is set to the `$attach_dir` directory defined in **bbs.setup** so that all uploaded files are placed in this directory by **cgi-lib.pl**.

```
if ($allow_user_attachments eq "on") {
    $cgi_lib'maxdata = $maximum_attachment_size;
    $cgi_lib'writefiles = "$attach_dir";
}
```

The incoming form variables are read into the `%in` associative array using the `ReadParse` routine.

```
&ReadParse;
```

If an alternative setup file has been defined using the `setup` form variable, this new setup file is loaded by the BBS script. Any new variables in the alternative setup file override those that were set previously in **bbs.setup**.

```
$setup_file = $in{"setup"};
if ($setup_file ne  "") {
    require "$setup_file.setup";
}
```

Then the authentication library is read. The BBS uses the session information generated by the authentication library throughout this script.

```
require "$auth_lib/auth-lib.pl";
```

Reading the Form Variables

The next part of the script reads and sets up nearly all the form variables. `$forum`, the variable name of the current discussion forum, is read first. It corresponds to a name in the `@forum_variables` array from which other forum information is gathered. The `GetForumInfo` subroutine then translates the abbreviated forum name (`$forum`) to a more descriptive forum name (`$forum_name`) and the directory (`$forum_dir`) that the forum messages will be found in.

```
$forum = $in{"forum"};
($forum_name, $forum_dir) = &GetForumInfo($forum);
```

The current session is read from the `session` form variable. If `session` is blank, the BBS sets `$is_this_a_new_session` to `yes` as a reminder that this is the first time that the BBS script has been called. The form variable, `session`, will be carried throughout the rest of the user's interaction with the BBS. Next, `GetSessionInfo` is called in the authentication library to retrieve information about the user if it exists. The details of this function are discussed in Chapter 9.

```
$session = $in{"session"};
$is_this_a_new_session = "yes" if ($session eq "");

($session, $username, $group,
 $firstname, $lastname, $email) =
 &GetSessionInfo($session, "$bbs_script", *in);
```

Whenever a button is pressed on an HTML form on the BBS, its name becomes associated with the value of the caption on the button. In addition, the BBS allows the user to specify the submit buttons as either images or normal buttons. For example, `$reply_op` will have a value

assigned to it if a reply_op button was pressed on a form or if a reply_op image button was pressed on an HTML form.

When an image tag is used in a form to simulate a button, the value of x and y coordinates for the variable are set instead of the variable name by itself. For example, if reply_op is the name of the image button and it is pressed, then the form variables reply_op.x and reply_op.y will have values assigned to them instead of the name reply_op by itself. These x and y coordinates correspond to the location of the mouse on the button when it was clicked. We know that the image button was pressed if either coordinate has a value associated with it.

$reply_op is assigned a value if the current operation is a reply to a currently read message. $post_op is assigned a value if the current operation being performed by the user is an original post to the BBS.

```
$reply_op = $in{"reply_op"};
$reply_op = "on" if ($in{"reply_op.x"} ne "");
$post_op = $in{"post_op"};
$post_op = "on" if ($in{"post_op.x"} ne "");
```

$create_message_op contains a value if the user has just submitted a message for posting.

```
$create_message_op = $in{"create_message_op"};
```

$read contains a value if the user is reading a message. The value of $read is the message file to read.

```
$read = $in{"read"};
```

$first_date, $last_date, $keywords, $exact_match, $first_days_old, and $last_days_old are form variables that the user can use to narrow the scope of the messages to be listed on the main forum page.

If $keywords are specified, all the messages displayed must contain those keywords in order to be seen. If $exact_match is on, the keywords must match whole words within the messages. For example, if a user searches for “ram” within a message, messages with words such as “ram,”

"cram," and others with "ram" inside them will return a positive match. However, with `exact_match` equal to `on`, only messages that contain the whole word "ram" will return a positive match.

`$first_date` and `$last_date` is a date range within which posting dates of the messages must fall if they are to be displayed to the user. `$first_days_old` and `$last_days_old` are similar to the previous two variables except that the age range of the posts is defined by "number of days old" rather than the explicit dates on which they were posted. These variables are discussed in more detail in the "Installation and Usage" section.

```
$first_date = $in{"first_date"};
$last_date = $in{"last_date"};
$keywords = $in{"keywords"};
$exact_match = $in{"exact_match"};
$first_days_old = $in{"first_days_old"};
$last_days_old = $in{"last_days_old"};
```

`$create_message_error` is a message that prints when the message list in a forum is printed if the previous attempt to create a message by a user has failed. One of the most common reasons for a failure is that the user left off a field that was specified as required in **bbs.setup**. For example, if the first name field is required and the user did not enter a first name, the post will fail. The message list in the forum is printed again, along with the error message detailing what happened in `$create_msg_error`.

```
$create_msg_error = "";
```

`$use_last_read` and `$last_read` are used if authentication is turned on and if the user wishes to see only new messages displayed in the forum list. `$use_last_read` is `on` if the user wants to use the "view only new messages" feature. `$last_read` is the sequence number of the user's last read message, which is compared against the sequence numbers of the current message list. If a sequence number of a message is higher than `$last_read`, we know that the message is "new."

```
$use_last_read = $in{"use_last_read"};
$last_read = $in{"last_read"};
```

Perform the BBS Operations

If `$reply_op` or `$post_op` has a value associated with it, it means that the user is either replying to a message or posting a new one, respectively. The `PrintPostOrReplyPage` function is called to print the HTML form for creating a message.

```
if (($reply_op ne "") || ($post_op ne ""))
{
    &PrintPostOrReplyPage;
}
```

If `$create_message_op` has a value, it means that a post or reply has been submitted to the BBS script for posting. In this case, the `CreatePosting` subroutine is called.

```
elsif ($create_message_op ne "")
{
    &CreatePosting;
}
```

If `$read` has a value associated with it, the BBS calls the `ReadMessage` subroutine to display the message. `$read` is a form variable—part of the URL hypertext link to each individual message in the message listing that is displayed to users when they first go into a discussion forum. For each hypertext link, `$read` corresponds to the message number to be read.

```
elsif ($read ne "")
{
    &ReadMessage($read);
}
```

Finally, as a catch-all, when `$forum` has a forum name associated with it, the list of messages is listed by the `PrintForumPage` procedure.

```
elsif ($forum ne "")
{
    &PrintForumPage;
}
```

The ReadMessage Subroutine

The ReadMessage subroutine takes a message filename as a parameter and displays it to the user as an HTML page.

```
sub ReadMessage
{
    local($message) = @_;
```

$poster_firstname, $poster_lastname, $poster_email, $post_date_time, $post_subject, and $post_options in the message header are declared local and are retrieved from the message file using the GetMessageHeader routine.

```
    local($poster_firstname, $poster_lastname,
          $poster_email,$post_date_time,
          $post_subject, $post_options) =
          &GetMessageHeader("$forum_dir/$message");
```

Then the body of the message is assigned to the $post_message variable. Because the message header has already been read, the first six lines of the file are skipped with a quick for loop.

```
    open (MESSAGEFILE, "$forum_dir/$message") ||
     &CgiDie("Could Not Open Message File\n");
    for (1 .. 6) { <MESSAGEFILE>; } # Throwaway header
    $post_message = "";
    while (<MESSAGEFILE>)
    {
     $post_message .= $_;
    }
    close (MESSAGEFILE);
```

It may seem inefficient that we open the message file twice—once to read the header and again to read the body—instead of doing it in one subroutine. However, message headers are read several times in the WebBBS script, so encapsulating that part of the script in a subroutine allows the code to be more modular. A common modification to the WebBBS script is to add your own header fields or subtract ones you do not need. Encapsulating the header information into a subroutine makes it easier for another programmer to make changes to the message structure.

If `$no_html_images` is `on`, image-related HTML is filtered out of the message body and subject so that the HTML code, rather than the image, is displayed to the user.

```
if ($no_html_images eq "on")
{
    $post_message =~ s/<(IMG\s*SRC.*)>/&LT;$1&GT;/ig;
    $post_subject =~ s/<(IMG\s*SRC.*)>/&LT;$1&GT;/ig;
} # End of parsing out no images
```

If `$no_html` is `on`, all the HTML code in the subject and message is filtered. Figure 25.10 shows an example of a message when its HTML code is filtered so that it will not affect the Web browser.

```
if ($no_html eq "on")
{
    $post_message =~ s/<([^>]+)>/\&LT;$1&GT;/ig;
    $post_subject =~ s/<([^>]+)>/\&LT;$1&GT;/ig;
} # End of No html
```

Figure 25.10 *WebBBS messages with HTML tags filtered.*

If there are multiple newlines (with a separating carriage return) in the BBS message, we assume that they indicate the start of a new paragraph and HTMLize them by replacing them with a <P> HTML tag. If one newline remains after the previous replacement, we assume it marks a simple line break, which is implemented using the
 HTML tag.

```
$post_message =~ s/\n\r\n/<p>/g;
$post_message =~ s/\n/<br>/g;
```

The following part of the routine opens the forum directory and attempts to read all the reply filenames to the @files array. Remember from the "Installation and Usage" section that message filenames follow the format **[message number] – [message number replied to].msg**. The grep command is used to read only those filenames whose "message number replied to" is the current message number being read. This information is used to generate a list of replies to the currently read message.

```
opendir(FORUMDIR, "$forum_dir") ||
    &CgiDie("Could not open $forum_dir directory\n");
$message_number = substr($message,0,6);
@files = sort(grep(/.......$message_number\.msg$/,
         readdir(FORUMDIR)));
closedir(FORUMDIR);
```

ReadMessageFields is called to create a URL listing of all the variables that must be passed from screen to screen on the BBS. This listing is placed into $message_url.

```
$message_url = &ReadMessageFields;
```

Next, the $post_replies HTML code is generated. First , $post_replies is cleared. Then, for every reply file in the @files array, the message header is read using the GetMessageHeader function, and a hypertext reference is generated and placed in $post_replies. At the end of the foreach loop, $post_replies will have all the message replies listed in HTML format.

```
$post_replies = "";
foreach (@files) {
    ($reply_firstname, $reply_lastname, $reply_email,
     $reply_date_time, $reply_subject,
```

```
     $reply_options) =
     &GetMessageHeader("$forum_dir/$_");
    $post_replies .=
      qq!<A HREF="$bbs_script?forum=$forum!;
    $post_replies .=
      qq!&read=$_&$message_url">!;
    $post_replies .=
     " $reply_subject " .
        "(Modified: $reply_date_time)</A><BR>\n";
}
```

The `$attach_file` filename is calculated by taking the message name minus the **.msg** extension and adding a new **.attach** extension. `$post_attach_html` is cleared and eventually is used to store the hypertext reference to an attached file if one exists.

```
$attach_file = substr($message,0,13) . ".attach";
$post_attach_html = "";
```

The `-e` operator is used to test whether the `$attach_file` exists. If it does, this file is opened and parsed to find out the name of the true attached file. The **[message name].attach** file contains only a descriptive reference to the uploaded file.

```
if (-e "$forum_dir/$attach_file") {
    open(ATTACHFILE, "$forum_dir/$attach_file") ||
     &CgiDie("Could Not Open $forum_dir/$attach_file\n");
    chop($attach_info = <ATTACHFILE>);

    ($post_attachment, $post_attachment_filename) =
      split(/\|/, $attach_info);
```

When the attached file has been found, a hypertext reference is generated to it in the `$post_attach_html` variable.

```
    $post_attach_html =
     qq!<BR><B>Attached File:</B> ! .
        qq!<A HREF="$attach_url/$post_attachment">
         $post_attachment_filename</A><BR>!;
    close (ATTACHFILE);
}
```

Finally, the **bbs_html_read_message.pl** library is called to print the full HTML related to the reading of a message. Figure 25.11 shows an example of a message that is displayed with replies and an attachment.

```
require "bbs_html_read_message.pl";

} # End of ReadMessage
```

Figure 25.11 *WebBBS message with replies and an attachment.*

The PrintForumPage Subroutine

The `PrintForumPage` subroutine displays the list of messages for the current forum in a hierarchical, threaded manner. There are no parameters to this routine other than the global form and setup variables that have been set previously. `$x` is set as a local variable and is used only to store temporary values.

```
sub PrintForumPage
{
    local($x);
```

The first step in displaying the messages is to read all the message files in the **$forum_dir** directory. The `grep` command is used to make sure that only files that end in **.msg** are listed. In addition, the `sort` command is used to sort the files in order. Because leading zeros are used to pad the six-digit message numbers, the messages are naturally sorted in ascending nu0meric order. The highest message number (`$high_number`) is set to the sequence number (the first six digits) of the last element of the `@files` array. `$low_number` is set to the first six digits of the first element of the `@files` array. These values are used later to determine how the messages relate to one another.

```
    opendir(FORUMDIR, "$forum_dir") ||
     &CgiDie("Could Not Open Forum Directory\n");
    @files = sort(grep(/.*msg$/,readdir(FORUMDIR)));
    closedir(FORUMDIR);
    $high_number = substr($files[@files - 1],0,6);
    $low_number = substr($files[0],0,6);
```

`PruneOldMessages` is called to delete old messages based on the parameters set in the BBS setup file.

```
       &PruneOldMessages($forum_dir, *files);
```

If the BBS setup file has defined `$display_only_new_message` as `on` and if the script has not yet determined the `$last_read` variable for the user, the `GetUserLastRead` function is called to retrieve this information. If `$last_read` is nothing, then it is set to 0 by default.

```
    if ($display_only_new_messages eq "on"
     && $last_read eq ""
     && $use_last_read eq "on") {
     $last_read =
          &GetUserLastRead($forum_dir,
                           $username, $session,
                           $high_number);
    }
```

```
    $last_read = 0 if ($last_read eq "");
```

The PruneFileList function is called to take the files in @files and delete all the messages that do not satisfy the filtering criteria that the user has set up previously.

```
&PruneFileList(*files, $last_read, $first_date,
               $last_date, $first_days_old,
               $last_days_old, $keywords,
               $exact_match, $forum_dir);
```

Now the heart of this subroutine processes the messages into HTML hypertext links along with brief descriptions and titles. Initially, $message_html and the @threads array are cleared.

```
$message_html = "";
@threads = ();
```

The MakeThreadList routine is called once for each thread in the @files message list. Each time MakeThreadList is called, all the message numbers that correspond to a message thread are deleted from the @files array and are moved into the @threads array in a way that is structured to make the threads easy to print as a hierarchical tree. By "tree," we mean that original posts are at the top level, replies to posts are at the second level, replies to replies are at the third level, and so on.

```
while (@files > 0)
     {
     push(@threads,&MakeThreadList(*files));
     }
```

After the list of threaded messages has been processed, the HTML must be generated. $ul_count is initialized to zero and is used to keep track of how deep the HTML indentations have gone. $prev_level is set to –1 initially and is used to determine which level of the thread we are currently on. Whenever we go up a level, a <UL> tag is printed for indenting, and whenever we go back down a level, the </UL> closure tag is printed to reverse the indentation.

```
$ul_count = 0;
$prev_level = -1;
```

The `foreach` loop goes through each of the messages stored in the `@threads` array.

```
foreach $x (@threads) {
```

The current level of the thread, the message filename, and the date of the newest message in the thread are `split` into the `$level`, `$messagefile`, and `$thread_date` variables.

```
($level,$messagefile, $thread_date) =
   split(/\|/,$x);
```

If `$level` is greater than the previous level and if `$level` is greater than `$display_thread_depth`, then `$level` is set back to `$prev_level`. This is because you set `$display_thread_depth` in the BBS setup file to indicate the deepest level of indentation you want the messages to be listed in. By setting the level to what it was previously whenever it goes beyond the `$display_thread_depth`, the script avoids indenting any further.

```
if ($level > $prev_level &&
 $level > $display_thread_depth) {
 $level = $prev_level;
}
```

If `$level` is greater than `$prev_level`, the `$ul_count` is incremented and an indentation tag (`<UL>`) is added to `$message_html`. If `$level` is less than `$prev_level`, then `$ul_count` is decremented and a `</UL>` closure tag is generated for every level of difference. Although messages tend to increase in level gradually, messages may decrease in level abruptly if the next message after a deeply nested thread is actually an original post and is not replying to anything previously posted.

```
if ($level > $prev_level) {
 $ul_count++;
 $message_html .= "<UL>\n";
```

```
    } elsif ($level < $prev_level) {
     for (1 .. ($prev_level - $level)) {
         $ul_count—;
         $message_html .= "</UL>\n";
       }
    }
```

If `$level` is simply equal to the previous level, then a line break HTML code (`<BR>`) is generated. If `$use_list_element` is `on`, a list button is generated using the HTML `<LI>` tag. `$use_list_element` is defined in the BBS setup file.

```
    if ($level == $prev_level) {
     if ($use_list_element ne "on"
         || $level == 1) {
         $message_html .= "<br>";
     }
     $message_html .= "\n";
    }
    if ($level > 1 && $use_list_element eq "on") {
     $message_html .= "<LI>";
    }
```

The header of the post is read using `GetMessageHeader`.

```
($poster_firstname, $poster_lastname, $poster_email,
 $post_date, $post_subject, $post_options) =
    &GetMessageHeader("$forum_dir/$messagefile");
```

The form variables that need to passed using hypertext links to the BBS script are generated using `ReadMessageFields`.

```
$message_url = &ReadMessageFields;
```

For each message, `$message_html` has a hypertext link generated that contains a call to **bbs_forum.cgi**. The parameters are set so that the particular message number is displayed if the user clicks the resulting hyperlink.

```
$message_html .=
  qq!<A HREF="$bbs_script?forum=$forum!;
$message_html .=
```

```
  qq!&read=$messagefile&$message_url">\n!;
$message_html .=
  " $post_subject ($post_date)";
```

Here is an example of a generated hyperlink from the previous code:

```
<A HREF="bbs_forum.cgi?forum=open&read=000001-000000.msg">
```

If we are at the top level of a thread and the date of the message is different from the date of the newest message in the thread (`$thread_date`), then an additional note lets the user know the most recent date that the thread was modified.

```
if ($level == 1 && $thread_date ne $post_date)
{
    $message_html .=
      " (Thread Modified:$thread_date)";
}
```

Then the end of the HTML hypertext reference is generated.

```
$message_html .= "</A>";
```

Before the script loops back to the next element of the `@threads` array, `$prev_level` is set to the current `$level` value.

```
    $prev_level = $level;
} # End of foreach thread
```

If all the messages have been processed and not enough `</UL>` tags have been generated, the rest are added to the `$message_html` variable.

```
$message_html .= "\n";
for (1..$ul_count) {
    $message_html .= "</UL>\n";
}
```

Finally, the script calls the **bbs_html_forum.pl** library to print the full HTML form listing all the messages in the forum. Figure 25.1 shows an example of the output from **bbs_html_forum.pl**.

```
require "bbs_html_forum.pl";

} # end of PrintForumPage
```

The PrintPostOrReplyPage Subroutine

`PrintPostOrReplyPage` prints the HTML code for the screen where a user can enter a new message. It gets its information from global variables that have already been set on the basis of the incoming form variables and the BBS setup file.

The most important form variables that this subroutine looks at are `$reply_to_message`, `$email_reply`, and `$post_subject`. These variables are relevant only if the HTML form is printing a reply type of message input for the user. The `$reply_to_message` is the number of the message the user is replying to. This message number is needed in order to allow users to "quote" the previous message in their post. `$email_reply` is the E-mail address of the user who posted the message in case he or she has chosen to receive automatic E-mail replies. `$post_subject` is the previous post subject so that "RE:" can be appended to it as the new default subject in the case of a reply. An example of a "reply" HTML page is shown in Figure 25.12.

```
sub PrintPostOrReplyPage
{
```

Several variables are defined as local to the subroutine and will be used at various points. `$options` is used to figure out whether the message being replied to has an E-mail option embedded in the header. `$reply_to_message`, `$email_reply`, and `$post_subject` were discussed in the previous paragraph. `$title` and `$header` are used as variables containing HTML code for the title and header of the HTML form.

```
    local($options, $post_subject);
    local($reply_to_message, $email_reply);
    local($title, $header);
    local($email_tag, $reply_to_email);
    $reply_to_message = "";
    $email_reply = "";
    $title = "Post A Message";
    $header = "Posting Message To $forum_name";
```

Figure 25.12 *Example of a reply screen.*

If the operation being performed by the user is a reply to a previously posted message, then the following code sets up the HTML posting form with the previous message's information.

```
if ($reply_op ne "")
{
```

`$reply_to_message` and `$email_reply` are initialized with their values from the form. In addition, the `$title` and `$header` of the HTML form are reassigned so that the user will be told that this is a reply rather than an original post HTML form.

```
$reply_to_message = $in{"reply_to_message"};
$email_reply = $in{"email_reply"};
```

```
$title = "Reply To A Message";
$header =
 "Replying To A Message In $forum_name";
```

Initially, `$post_message` is cleared. Then the file for the message being replied to is opened and read to memory. The various fields—such as first name, last name, date and time of the post, and options—are read to variables that will be used later to generate the header of a quoted message.

```
$post_message = "";
open (REPLYFILE,
   "$forum_dir/$reply_to_message") ||
    &CgiDie("Could not open reply message");
chop($post_first_name = <REPLYFILE>);
chop($post_last_name = <REPLYFILE>);
<REPLYFILE>;
chop($post_date = <REPLYFILE>);
<REPLYFILE>;
chop($options = <REPLYFILE>);
if ($options =~ /^options:/) {
 $options = substr($options,8);
 ($email_tag,$reply_to_email) =
     split(/:/,$options);
}
```

The body of the message is read to the `$post_message` variable.

```
while (<REPLYFILE>) {
 $post_message .= $_;
}
```

When the message is read, the beginning of the `$post_message` buffer (`^`) is preceded with two greater-than symbols (`>>`). In addition, each newline or carriage return combination in the `$post_message` buffer has the double greater-than symbols (`>>`) added to it. This is done so that the quoted text appears different from the user's own comments by default. The convention of preceding reply lines with "`>>`" is an Internet custom followed by many programs.

```
$post_message =~ s/^/>>/g;
$post_message =~ s/\r/\r>>/g;
$post_message =~ s/\n/\n>>/g;
```

In addition, the poster's information is added to the beginning of the $post_message variable as a header giving credit to the original author of the quoted post by default.

```
        $post_message =
         "$post_first_name $post_last_name" .
             " on $post_date said:\n\n"
             . $post_message;
        close (REPLYFILE);
```

$post_subject is read from the form variable, and "Re: " (short for "regarding") is added to the beginning of it if the subject does not already begin with this prefix.

```
        $post_subject = $in{"post_subject"};
        $post_subject = "Re: $post_subject"
         if !($post_subject =~ /^Re:/i);
    }
```

$post_date_time is set equal to a formatted date and time generated by the GetDateAndTime subroutine.

```
    $post_date_time = &GetDateAndTime;
```

The <INPUT> fields are generated based on the user information that was given previously.

```
    $post_first_name_field = qq!<INPUT TYPE=text
     NAME=form_firstname VALUE="$firstname"
     SIZE=40 MAXLENGTH=50>!;
    $post_last_name_field = qq!<INPUT TYPE=text
     NAME=form_lastname VALUE="$lastname"
        SIZE=40 MAXLENGTH=50>!;
    $post_email_field = qq!<INPUT TYPE=text
     NAME=form_email VALUE="$email"
        SIZE=40 MAXLENGTH=50>!;
```

In the BBS setup file, you can define certain header fields to be forced values. If the values are forced, then instead of giving the user an <INPUT TYPE=TEXT> field, an <INPUT TYPE=HIDDEN> field is generated and the user sees only a display of header information and cannot change it. This is

done for $force_first_name, $force_last_name, and $force_email. This feature works only if authentication is turned on in the BBS setup file in addition to these forced variables.

```
if ($force_first_name eq "on"
 && $firstname ne "") {
 $post_first_name_field =
 qq!<INPUT TYPE=hidden
 NAME=form_firstname VALUE="$firstname">!;
 $post_first_name_field .= "$firstname";
}

if ($force_last_name eq "on"
 && $lastname ne "") {
 $post_last_name_field =
 qq!<INPUT TYPE=hidden
 NAME=form_lastname VALUE="$lastname">!;
 $post_last_name_field .= "$lastname";
}

if ($force_email eq "on"
 && $email ne "") {
 $post_email_field =
    qq!<INPUT TYPE=hidden
 NAME=form_email VALUE="$email">!;
 $post_email_field .= "$email";
}
```

If the $allow_reply_email variable is set to on in the BBS setup file, $post_want_email contains HTML code for a check box in which users can choose to have replies E-mailed to them.

```
$post_want_email = "";
if ($allow_reply_email eq "on") {
 $post_want_email =
     "<BR><INPUT TYPE=CHECKBOX" .
     " NAME=post_want_email>" .
        "Check Here If You Want Replies " .
     "Emailed To You Automatically<BR>";
}
```

If $allow_user_attachments is set to on in the BBS setup file, $post_attachment contains HTML code for the file upload input box.

```
    $post_attachment = "";
    if ($allow_user_attachments eq "on") {
     $post_attachment =
         "<P>Attach A File Here:
<INPUT TYPE=FILE NAME=post_attachment><BR>";
    }
```

Finally, **bbs_html_create_message.pl** is loaded to print the HTML for the Create Message screen.

```
    require "bbs_html_create_message.pl";

} # End of PostOrReplyPage
```

The CreatePosting Subroutine

The `CreatePosting` subroutine takes the information that a user has entered on a Post Message HTML form and posts it as a message file in the forum directory. After the message is posted, this function calls the subroutine that prints the message list. The form variables `form_firstname`, `form_lastname`, `form_email`, `form_subject`, `form_message`, `reply_to_message`, `reply_to_email`, and `post_want_email` make up the contents of the post information that is written to the message file.

```
sub CreatePosting
{
```

`$create_error` is declared as local. If there is any problem in posting the message to a file, `$create_error` will contain a positive numeric value. This triggers an error message that is sent to the routine that prints the forum's message list so that the user is notified of the problem.

```
    local ($create_error);
```

All the form variables related to the header of the message are read to variables.

```
    $form_firstname = $in{"form_firstname"};
    $form_lastname = $in{"form_lastname"};
```

```
$form_email = $in{"form_email"};
$form_subject = $in{"form_subject"};
```

Then newlines are stripped out as a precaution against multiline values. The header values should consist only of single lines.

```
$form_firstname =~ s/\n//g;
$form_lastname =~ s/\n//g;
$form_email =~ s/\n//g;
$form_subject =~ s/\n//g;
```

`$form_message` is set to the body of the message that was posted.

```
$form_message = $in{"form_message"};
```

`$reply_to_message` is the message number that is being replied to if the user is posting a reply. The message reply number is needed because when a message filename is generated, the second six-digit number is the message number that is being replied to. If the message number is less than 1, the message to reply to is set to six zeros, indicating that the message is not a reply to anything.

```
$reply_to_message = $in{"reply_to_message"};
if ($reply_to_message < 1)
{
   $reply_to_message = "000000";
} else {
   $reply_to_message =
    substr($reply_to_message,0,6);
}
```

`$reply_to_email` is the E-mail address of the person who posted the message that is being replied to if the message being created is a reply rather than an original post. `$post_date_time` is set to the current date and time returned by the `GetDateAndTime` function.

```
$reply_to_email = $in{"reply_to_email"};
$post_date_time = &GetDateAndTime;
```

`$form_options` consists of the options that the user has chosen for the post that is being created. The only option currently implemented is the

automatic E-mail replies option. If `$post_want_email` is `on` or if `$force_reply_email` is set to `on` in the setup file, then the user's E-mail address is added to the options tag to indicate that the user wants automatic E-mail replies.

```
    $form_options = "";
    $post_want_email = $in{"post_want_email"};
    if ($post_want_email eq "on"
     || $force_reply_email eq "on") {
     $form_options = "email:$form_email";
    }
```

`$create_error` is initialized to zero. This indicates that no error has occurred yet.

```
    $create_error = 0;
```

If any of the required fields has not been entered by the user, `$create_error` is set to 1 and `$create_msg_error` is filled with the appropriate error message. The required fields are determined in **bbs.setup** by the `$require_subject`, `$require_first_name`, `$require_last_name`, and `$require_email` variables.

```
if ($require_subject eq "on" &&
    $form_subject eq "") {
    $create_error = 1;
    $create_msg_error .=
      "You Did Not Enter A Subject.</H2>";
}
if ($require_first_name eq "on" &&
    $form_firstname eq "") {
    $create_error = 1;
    $create_msg_error .=
      "You Did Not Enter Your First Name.</H2>";
}
if ($require_last_name eq "on" &&
    $form_lastname eq "") {
    $create_error = 1;
    $create_msg_error .=
      "You Did Not Enter Your Last Name.</H2>";
}
if ($require_email eq "on" &&
    $form_email eq "") {
    $create_error = 1;
    $create_msg_error .=
```

```
      "You Did Not Enter An Email Address.</H2>";
}
```

If `$create_error` is 1, then a header indicating an error posting to the BBS is appended to the beginning of `$create_msg_error`.

```
if ($create_error == 1) {
$create_msg_error = "<HR><H2>Error Posting To BBS. " .
  $create_msg_error;
}
```

If `$create_error` is not equal to 1, then the posting of the message can proceed.

```
if ($create_error != 1) {
```

`$whole_msg` is set equal to all the header fields plus the body of the message stored in `$form_message`.

```
    $whole_msg = "";
    $whole_msg .= "$form_firstname\n";
    $whole_msg .= "$form_lastname\n";
    $whole_msg .= "$form_email\n";
    $whole_msg .= "$post_date_time\n";
    $whole_msg .= "$form_subject\n";
    $whole_msg .= "options:$form_options\n";
    $whole_msg .= "$form_message\n";
```

The message files in the forum directories are read to the `@files` array. As before, the `grep` command is used to retrieve only the filenames corresponding to messages, and the `sort` command makes sure that the `@files` array is in numerically ascending order.

```
opendir(FORUMDIR, "$forum_dir") ||
    &CgiDie("Couldn't Open $forum_dir");
@files = sort(grep(/.*msg$/,readdir(FORUMDIR)));
closedir(FORUMDIR);
```

`$high_number` is set to the highest message number value in the `@files` array. Because the array is sorted, this is the last element of the array. `@files` minus 1 is the highest element number in the `@files` array, because arrays

count from zero to the number of elements in the array minus 1. Once the high message number has been retrieved, the `@files` array is cleared. Then `$high_number` is incremented by 1. This determines the sequence number for the message filename that will be created later in the routine.

```
$high_number = substr($files[@files - 1],0,6);
@files = ();
$high_number++;
```

The `$high_number` is formatted to a length of six using the `sprintf` function (`%6d`). Then the leading spaces are converted to zeros using the `tr` function. If there is no `$high_number` (equal to 000000), then `$high_number` is set to `"000001"`. Finally, the `$message_name` filename is generated by joining the high number and the `$reply_to_message` variable with a hyphen (-).

```
$high_number = sprintf("%6d",$high_number);
$high_number =~ tr/ /0/;
    $high_number = "000001"
     if ($high_number eq "000000");
$message_name = "$high_number-$reply_to_message";
```

The file is created with an **.msg** extension. `$whole_msg` is written to the file, and then it is closed.

```
open(WRITEMSG, ">$forum_dir/$message_name.msg") ||
    &CgiDie("Couldn't open $message_name.msg");
print WRITEMSG $whole_msg;
close (WRITEMSG);
```

`$post_attachment` is set equal to the `post_attachment` form variable that contains the name randomly assigned by **cgi-lib.pl** to the file that was uploaded. `$post_attachment_filename` is set to the value that **cgi-lib.pl** retrieved from the Web browser as the real filename of the uploaded file. The real filenames are stored by **cgi-lib.pl** in the `%incfn` associated array apart from the `%in` associative array for normal form variables. This is explained in more detail in Chapter 5.

```
$post_attachment = $in{"post_attachment"};
$post_attachment_filename =
    $incfn{"post_attachment"};
```

Any hexadecimal values in the filename are parsed into characters using a regular expression. Basically, the regular expression filters on any two-digit hexadecimal digits (A-Fa-f0-9). The result of the filter is placed inside the `$1` variable by Perl, and the `pack` command is used to convert the two-digit number to an ASCII character.

```
$post_attachment_filename =~
    s/%([A-Fa-f0-9]{2})/pack("c",hex($1))/ge;
```

If a `$post_attachment_filename` exists, the uploaded file is renamed to a filename that represents the message that it belongs to except that it has a **.bin** extension instead of an **.msg** extension. The **.bin** extension is used because, by default, most Web browsers are configured to download **.bin** files—and not to display their contents—when they are selected by a hypertext link.

```
    if ($post_attachment_filename ne "") {
     rename($post_attachment,
            "$attach_dir/$forum-$message_name.bin");
```

A descriptive message file with an **.attach** extension is created to keep track of the filename of the originally uploaded file and the new filename assigned to it by the BBS.

```
     open(WRITEATTACH,
          ">$forum_dir/$message_name.attach") ||
           &CgiDie("Could Not Open Attachment\n");
     print WRITEATTACH
         "$forum-$message_name.bin" .
          "|$post_attachment_filename\n";
     close(WRITEATTACH);
```

If there is no postattachment filename, the attachment is removed.

```
    } else {
     unlink("$post_attachment");
    }
```

If the message being posted is a reply to a previous message and if an automatic reply must be generated in E-mail, then **mail-lib.pl** is loaded to

send the mail. The reply subject is set to a description telling the user that a post was made on the BBS, and the body of the message is sent as part of the body of the E-mail.

```
$reply_to_email = $in{"reply_to_email"};
    if ($reply_to_email ne "" &&
     $send_reply_email eq "on") {
     require "$lib/mail-lib.pl";
     $reply_subject =
         "Reply to your $forum_name message.";
     &send_mail($from_email, $reply_to_email,
             $reply_subject,
             "The Message:\n\n" . $form_message);
    } # End of reply_to_email
} # end of if $create_error == 1
```

Finally, the message list of the forum is displayed using the `PrintForumPage` function.

```
&PrintForumPage;
} # End of CreatePosting
```

The PruneFileList Subroutine

`PruneFileList` takes the message filenames in a forum directory as an array and removes any files that do not match certain user-defined criteria. The variables related to these criteria are `$last_read`, `$first_date`, `$last_date`, `$first_days_old`, `$last_days_old`, `$keywords`, and `$exact_match`. These variables correspond to the equivalent form variables that are used to filter the message list in the **bbs_entrance.cgi** script. These variables were discussed previously in the "Installation and Usage" and "Design Discussion" sections. `$forum_dir` is also passed to the routine so that the script knows which directory to search for files.

```
sub PruneFileList
{
    local(*files, $last_read, $first_date, $last_date,
       $first_days_old, $last_days_old, $keywords,
       $exact_match, $forum_dir) = @_;
```

`$x`, `$filename`, `$month`, `$day`, `$year`, `$comp_date`, and `$file_date`, which will be used in the filtering process later, are declared local.

```
local($x, $filename);
local($month, $day, $year, $comp_date);
local($file_date);
```

If a keyword search is being performed, the keywords are `split` into separate words into an array called `@keyword_list`. The regular expression `\s` matches on whitespace characters that might separate the keywords in the `$keywords` variable.

```
@keyword_list = split(/\s+/,$keywords);
```

Each file in the `@files` message filename is processed using the `for` loop. There are four different reasons that a filename would be removed from the list of messages displayed to the user: the user has already read the message, the age of the message does not fit the specified range of days, the date on which the message was posted does not fit the specified range of dates, and the message does not contain the specified keywords. If the message does not satisfy all the requirements, it is removed from the `@files` array and is not displayed to the user.

```
for ($x = @files; $x > 0; $x—)
{
```

In case 1, the message was already read. If the `$last_read` message number variable is greater than zero, if the current message number is less than `$last_read`, and if `$display_only_new_messages` has been set to `on` in the BBS setup file, then the filename is removed from the list of messages to display.

```
if ($last_read > 0
        && substr($files[$x-1],0,6) <= $last_read
        && $display_only_new_messages eq "on")
       {
        &RemoveElement(*files,$x-1);
        next;
       }
```

Remember that arrays start counting at zero. Thus, the elements of the `@files` array are referenced by `$x` minus 1, because `$x` counts from 1 to the number of elements in the array.

In case 2, the message does not fit the days-old age range. Before we process the age of the messages, the full path and filename are placed in the `$filename` variable. We do this so that the BBS script can check the age of the post by looking at the individual message file.

```
$filename = "$forum_dir/$files[$x - 1]";
```

If the age of the filename (determined with the `-M` operator) is greater than `$first_days_old`, the filename is removed from the list of messages to display.

```
if (($first_days_old ne "")
 && ((-M $filename) > $first_days_old)) {
 &RemoveElement(*files,$x-1);
 next;
}
```

Although `-M` technically checks the modification date of the file, for the purpose of the `bbs_forum.cgi` script, the value returned by `-M` effectively represents the age of the file.

If the age of the filename (`-M`) is less than `$last_days_old`, the filename is removed from the list of messages to display.

```
if (($last_days_old ne "")
 && ((-M $filename) < $last_days_old)) {
 &RemoveElement(*files, $x-1);
 next;
}
```

In case 3, the message does not fit the date range. To compare the dates of the files, the script rearranges the date range to make it more conducive to comparison. Normally, dates are stored in an MM/DD/YY format. However, it is impossible to compare such a date numerically with another date to see which one is greater. Instead, the dates are rearranged into a number in which the year is the first part, the month is the second part, and the day is last: a YYMMDD format. These newly formatted dates can be compared using normal numerical comparison methods.

Let's look at some example dates: 12/01/95, 01/01/96, and 11/15/96. If the slashes were removed, these dates would form the following numbers: 120195, 010196, and 111596. Although 01/01/96 is a greater date than 12/01/95, the number 120195 is greater than 010196. Thus, doing a numerical comparison does not work when the dates are formatted conventionally.

Reformatting these dates into a YYMMDD format makes them into 951201, 960101, and 961115. Clearly, a numerical comparison now yields the correct answer when the dates are compared. For example, 960101 is greater than 951201, which yields the correct answer that 01/01/96 is a greater date than 12/01/95. The BBS script converts dates to the YYMMDD numerical format for this reason. The following pieces of code perform this conversion.

```
if ($first_date ne ""
 || $last_date ne "") {
```

`$month`, `$day`, and `$year` are `split` from the date in `$first_date`.

```
($month, $day, $year) =
    split(/\//, $first_date);
```

If `$month` and `$day` are not two digits long, they are padded with a leading zero.

```
$month = "0" . $month
    if (length($month) < 2);
$day = "0" . $day
    if (length($day) < 2);
```

If the `$year` is a two-digit one, it is converted to a four-digit number so that when we reach the year 2000, 00 will not seem smaller than 99 after they are converted to 2000 and 1999, respectively. If a year is less than 95, it is basically converted to a 21st-century year (20xx) by adding 2000 to it. Otherwise, the year is considered a 20th-century year (19xx).

```
if ($year > 95 && $year < 1900) {
    $year += 1900;
```

```
}
if ($year < 1900) {
    $year += 2000;
}
```

Finally, the date is rearranged into the YYMMDD format and is assigned to `$comp_first_date`.

```
$comp_first_date = $year . $month . $day;
```

The same process is applied to `$last_date`. The YYMMDD format is placed in the `$comp_last_date` variable.

```
($month, $day, $year) =
    split(/\//, $last_date);
$month = "0" . $month
    if (length($month) < 2);
$day = "0" . $day if (length($day) < 2);
if ($year > 50 && $year < 1900) {
    $year += 1900;
}
if ($year < 1900) {
    $year += 2000;
}
$comp_last_date = $year . $month . $day;
```

The actual date of the file is retrieved by using Perl's `stat` command, which returns an array of elements corresponding to different pieces of information about the file. The 10th piece of information is the modification date and time of the file. This means that the script references element number 9, because arrays start counting from zero.

```
$file_date = (stat($filename))[9];
```

The `localtime` function is used to pull out the day, month, and year for the `$file_date`. Then the same process is applied to this date that was performed on `$first_date` and `$last_date`. `$file_date` ends up in the YYMMDD format of `$comp_first_date` and `$comp_last_date`.

```
($day, $month, $year) =
    (localtime($file_date))[3,4,5];
```

```
$month++;
$month = "0" . $month
    if (length($month) < 2);
$day = "0" . $day if (length($day) < 2);
if ($year > 50 && $year < 1900) {
    $year += 1900;
}
if ($year < 1900) {
    $year += 2000;
}
$file_date = $year . $month . $day;
```

The next two pieces of code compare `$file_date` against `$comp_first_date` and `$comp_last_date`. If `$file_date` is not in this range of dates, the filename is removed from the `@files` array.

```
if ($first_date ne "") {
    if ($file_date < $comp_first_date) {
     &RemoveElement(*files, $x-1);
     next;
    }
} # End of first date

if ($last_date ne "") {
    if ($file_date > $comp_last_date) {
     &RemoveElement(*files, $x-1);
     next;
    }
} # End of last date

} # End of First or Last Date
```

In case 4, the message does not contain the keywords. If `$keywords` exist, the message file is opened and searched for keywords. `@not_found_words` is initialized to the list of keywords. For each line of the file, `@not_found_words` is checked to see whether any of the words exist on the line using the `FindKeywords` function. If a keyword is found, it is removed from the `@not_found_words` array.

```
  if ($keywords ne "") {
@not_found_words = @keyword_list;
open(SEARCHFILE, $filename);
while(<SEARCHFILE>) {
```

```
        $line = $_;
        &FindKeywords($exact_match, $line,
                *not_found_words);
    } # End of SEARCHFILE
    close (SEARCHFILE);
```

This keyword searching algorithm is basically the same one used in Chapter 16 about the keyword search engine.

If any words are still in the `@not_found_words` array, the file failed the keyword search and is removed from the `@files` array.

```
    if (@not_found_words > 0) {
        &RemoveElement(*files, $x - 1);
        next;
    }
      } # End of keywords
   } # End of for loop

} # End of PruneFileList
```

The FindKeywords Subroutine

The `FindKeywords` subroutine is the search routine that is called by the `PruneFileList` function. It accepts a line of a message file and the keywords to search for in that line. If a keyword is found, the routine removes it from the keyword array (`@not_found_words`). Thus, when the `@not_found_words` array no longer contains any elements, the script knows that all the keywords have been found in the message file.

```
sub FindKeywords
{
```

There are three parameters. The first, `$exact_match`, is equal to `on` if the type of pattern match we are doing is based on an exact one-to-one match of each letter in the keyword to each letter in a word contained in

the message. The second parameter, `$line`, is the line in the message that is currently being searched. The third parameter, `*not_found_words`, is a reference to the array `@not_found_words`, which contains a list of all the keywords not found so far. As keywords are found in the searched file, this array has its words removed. Thus, when the array is empty, we know that the message file contained all the keywords. In other words, there are no "not found words" if the search is successful.

```
local($exact_match, $line, *not_found_words) = @_;
local($x, $match_word);
```

If the exact match is on, the program matches all the words in the array by surrounding the keywords with `\b`. This means that the keyword must be surrounded by word boundaries to be a valid match. Thus, the keyword "the" would not match a word such as "there," because it is only part of a larger word.

```
if ($exact_match eq "on") {
 for ($x = @not_found_words; $x > 0; $x—) {
     $match_word = $not_found_words[$x - 1];
     if ($line =~ /\b$match_word\b/i) {
```

The `splice` routine cuts out the words if they are found in the search. The `splice` command is a Perl routine that accepts the original array, the element in the array to splice, the number of elements to splice, and a list or array to splice into the original array. Because we are leaving off the fourth parameter of the splice, the routine by default splices "nothing" into the array at the element number. This action deletes the element in one convenient little routine.

```
      splice(@not_found_words,$x - 1, 1);
     } # End of If
 } # End of For Loop
```

If the exact match is not on, the program finds a match if the letters in the keyword exist anywhere on the line, whether or not the keyword is part of a larger word. All the searches are case-insensitive, as indicated by the `i` after the slashes defining the search term.

```
    } else {
     for ($x = @not_found_words; $x > 0; $x—) {
         $match_word = $not_found_words[$x - 1];
         if ($line =~ /$match_word/i) {
          splice(@not_found_words,$x - 1, 1);
         } # End of If
     } # End of For Loop
    } # End of ELSE

} # End of FindKeywords
```

The GetUserLastRead Subroutine

The `GetUserLastRead` subroutine gets the user's last read message number. In addition, if this is the first time the session has been created, the user's last read message number is updated with the current high message number.

```
sub GetUserLastRead
{
```

`$forum_dir` (forum directory), `$username` (username of the user), `$session` (session ID), and `$high_number` (highest message number) are passed as local parameters to the routine. The `$last_read` and `$old_session` variables are declared local to `GetUserLastRead`.

```
    local($forum_dir, $username,
       $session, $high_number) = @_;
    local($last_read, $old_session);
```

[username].dat is used in each forum directory to keep track of the last read message of a user between sessions. If the user logs on to the BBS and if the BBS has been configured to allow users to read only new messages, the user's name, followed by a **.dat** extension, is created in each forum he or she reads. If the **[username].dat** file does not exist in the forum directory, then the last read message number is set to zero.

```
unless (-e "$forum_dir/$username.dat")
     {
     $last_read = 0;
     }
```

Otherwise, **[username].dat** is opened and the **$last_read** variable is read. In addition, the last session that was active when the file was created is read to the `$old_session` variable.

```
else
{
open (USERFILE, "$forum_dir/$username.dat") ||
    &CgiDie("Error Opening Userfile $username\n");
$last_read = <USERFILE>;
$old_session = <USERFILE>;
chop ($last_read);
chop($old_session);
close (USERFILE);
}
```

If the current session (`$session`) is not equal to the session ID that was used when **[username].dat** was created, the script knows that this is a new session; the file is rewritten with the new high message number and the current session ID. The current high message number then becomes the new last read message number the next time the user goes into the BBS script with a new session.

In other words, the first time a user logs in to a BBS discussion forum, the current last read message is retrieved from the user file and then the user file is updated with the highest message number. All subsequent accesses to the script within the same session will simply retrieve the last read message number from a form variable instead of accessing this file. Then, when the user logs in to the BBS again with a new session, the last read message is read again from the user file.

The reason the user file is updated immediately with the highest message number is that there is no guarantee that the user will return to the discussion forum after reading a message. Because the Web is connectionless, there is nothing to stop the user from going to another site on the Web. Thus, the user file is updated immediately with the high message number after the last read message number is read.

```
if ($session ne $old_session) {
 open (USERFILE, ">$forum_dir/$username.dat") ||
     &CgiDie("Error Opening Userfile $username\n");
 print USERFILE "$high_number\n";
```

```
     print USERFILE "$session\n";
     close (USERFILE);
    }
```

Finally, the `$last_read` message number is returned to the calling procedure.

```
$last_read;

} #End of GetUserLastRead
```

The GetDateAndTime Subroutine

`GetDateAndTime` returns a formatted string with the current date and time. The parts of the date and time returned from the `localtime` function are declared as local to the subroutine.

```
sub GetDateAndTime
{
    local ($sec, $min, $hour, $mday, $mon);
    local($year, $wday, $yday, $isdst);
    local ($ampm);
```

The `localtime(time)` function is called in Perl to get the current date and time values.

```
($sec, $min, $hour, $mday, $mon,
 $year, $wday, $yday, $isdst) =
    localtime(time);
```

Because months in the `localtime` function are numbered from zero to 11, we increment it by 1.

```
$mon++;
```

Next, the script determines whether we are in "AM" or "PM."

```
$ampm = "AM";
$ampm = "PM" if ($hour > 11);
$hour = $hour - 12 if ($hour > 12);
```

If the $min variable is only one digit, the script pads it with a leading zero.

```
if (length($min) == 1)
{
    $min = "0" . $min;
}
```

Finally, the formatted date is returned by the subroutine.

```
"$mon/$mday/$year $hour:$min $ampm";

} # End of GetDateAndTime
```

The GetMessageHeader Subroutine

GetMessageHeader takes a filename as a parameter and returns an array containing the values for all the header fields in the message file.

```
sub GetMessageHeader
{
    local($filename) = @_;
```

$poster_firstname, $poster_lastname, $poster_email, $post_date, $post_subject, and $post_options are declared local. Each variable corresponds to a field in the message header.

```
    local($poster_firstname, $poster_lastname,
       $poster_email, $post_date,
       $post_subject, $post_options);
```

The file is opened. If the open file routine is unsuccessful, CgiDie is called to output an error message to the user's Web browser and then exit.

```
    open (MESSAGEFILE, "$filename") ||
     &CgiDie("Could Not Open $filename hdr\n");
```

Each variable is read from the message file using the <MESSAGEFILE> command. By surrounding the file handle with brackets, Perl returns the

next line of the file. In addition, the resulting variables are chopped to get rid of the superfluous newline character that is read with every line in the message header.

```
chop($poster_firstname = <MESSAGEFILE>);
chop($poster_lastname = <MESSAGEFILE>);
chop($poster_email = <MESSAGEFILE>);
chop($post_date = <MESSAGEFILE>);
chop($post_subject = <MESSAGEFILE>);
chop($post_options = <MESSAGEFILE>);
```

The `options:` portion of the post options line in the header is stripped off using the following code. This is because the word "options" is superfluous. If there is an E-mail option, it will be evident from the rest of the options line. `substr($post_options, 8` returns everything except the first eight characters of `$post_options`. The message file is closed when all the header fields are read.

```
if ($post_options =~ /^options:/) {
 $post_options = substr($post_options,8);
}
close(MESSAGEFILE);
```

Finally, each of the header fields is returned as a separate element in an array.

```
($poster_firstname, $poster_lastname, $poster_email,
 $post_date, $post_subject, $post_options);
```

} # End of GetMessageHeader

The MakeThreadList Subroutine

The `MakeThreadList` subroutine takes the sorted list of messages stored in `@file_list` and returns an entire thread of messages that has been pulled out of the `@file_list`.

Why? Because `@file_list` merely stores messages in the order that they were posted. It does not show the relationship of the replies. It does not show that message number 3 may have been a reply to message number 1 and that message number 2 may be a post all by itself and not a reply to any other post.

Lets look at a simple example. A *thread* is a list of messages that are related by replies and have one post as the starting point. Let's start by taking a look at 10 sample message filenames (without the **.msg** extension). Remember, the fact that a message is a reply is shown in the second six-digit number.

```
000001-000000 (Message 1 Is An Original Message)
000002-000001 (Message 2 Replies To Message 1)
000003-000000 (Message 3 Is An Original Message)
000004-000002 (Message 4 Replies To Message 2)
000005-000001 (Message 5 Replies To Message 1)
000006-000000 (Message 6 Is An Original Message)
000007-000003 (Message 7 Replies To Message 3)
000008-000005 (Message 8 Replies To Message 5)
000009-000008 (Message 9 Replies To Message 8)
000010-000007 (Message 10 Replies To Message 7)
```

In this example, the messages appear in the order in which they were posted, but figuring out which messages are related to each other in separate groupings is difficult. `MakeThreadList` takes care of this task.

The best way to describe how `MakeThreadList` works is to discuss the basic algorithm. Imagine that `MakeThreadList` is given an array of the 10 filenames we were shown previously. The routine starts by looking at the highest message number (000010-000007). If it is a reply to a previous message, that message is examined (000007-000003); basically, any time a message has a reply, the routine goes to the replied message. In this case, message 3 is examined (000003-000000). Because the reply to message number 7 is all zeros, we know that message 3 is a top-level post. The routine stops going backward through the message list.

A neat side effect of the script starting to look for threads at the highest message number is that when the threads are retrieved, the threads naturally sort themselves so that the thread with the newest message appears first in the list.

Then 3 three is removed from the array and is placed into a threads list with a level number of 1 indicating that it is a top-level post. At this point, the message list looks like this:

```
000001-000000
000002-000001
000004-000002
000005-000001
000006-000000
000007-000003
000008-000005
000009-000008
000010-000007
```

And thread list consists of the following:

```
000003-000000 At Level 1
```

Then the script looks at all the replies to message 3. When it searches the file list, it finds 000007-000003 and places that in the thread list with a level of 2 (because it is a reply to an original post). The message list now looks like the following:

```
000001-000000
000002-000001
000004-000002
000005-000001
000006-000000
000008-000005
000009-000008
000010-000007
```

The thread list looks like this:

```
000003-000000 At Level 1
000007-000003 At Level 2
```

Because message 7 has been pulled out, the script looks at the file list for replies to message 7. At this point, message 10 is a reply and so is added to the thread list at level 3 (it is a reply to a reply). The message list now looks like this:

```
000001-000000
000002-000001
```

```
000004-000002
000005-000001
000006-000000
000008-000005
000009-000008
```

The thread list looks like this:

```
000003-000000 At Level 1
000007-000003 At Level 2
000010-000007 At Level 3
```

Now the script looks at the file list for replies to message 10. There are none, so the script goes back up one level (back to level 2) and looks for more replies to message 7. There are none, so the script goes back up one level (to level 1) and looks for more replies to message 3. There are none, and there are also no higher levels, so the thread is complete. The `MakeThreadList` routine ends and returns a complete thread to the BBS. The final ordered thread list remains the same:

```
000003-000000 At Level 1
000007-000003 At Level 2
000010-000007 At Level 3
```

The first example was a relatively simple hierarchy with only one reply branch. This next example is more complex. When `MakeThreadList` is called a second time with the previous pared-down file list, the routine looks at the highest message number (message 9) and sees that it is a reply to another message (message 8). Message 8 is seen as a reply to message 5, and message 5 is a reply to message 1 which has no successor. At this point, the routine places message 1 in the thread list. The message list now looks like this:

```
000002-000001
000004-000002
000005-000001
000006-000000
000008-000005
000009-000008
```

The thread list that we are starting again has just one message at level 1.

```
000001-000000 At Level 1
```

The routine then looks at the next reply to message number 1. The first reply is message 2. It is pulled into the thread list and is placed at level 2 because it is a reply to the original post (message 1). The message list now looks like this:

```
000004-000002
000005-000001
000006-000000
000008-000005
000009-000008
```

The thread list looks like this:

```
000001-000000 At Level 1
000002-000001 At Level 2
```

The routine then looks for replies to message 2. Message 4 fits this criterion. Thus, message 4 is placed in the thread list at level 3 (a reply to a reply). The message list now looks like this:

```
000005-000001
000006-000000
000008-000005
000009-000008
```

The thread list looks like this:

```
000001-000000 At Level 1
000002-000001 At Level 2
000004-000002 At Level 3
```

There are no replies to message 4, so the routine goes back to level 2 and checks for more replies to message 2. There are no more replies to message 2, so the routine goes back to level 1 and checks for more replies to message 1. Message number 5 is a match and is pulled from the message list and placed in the thread list at level 2 (because it is a reply to the

original post). The message list now looks like this:

```
000006-000000
000008-000005
000009-000008
```

The thread list looks like this:

```
000001-000000 At Level 1
000002-000001 At Level 2
000004-000002 At Level 3
000005-000001 At Level 2
```

Next, the BBS looks for replies to message 5. Message 8 qualifies, so it is placed in the thread list at level 3. Next, replies to message 8 are searched for, and message 9 yields a match. Thus, message 9 is placed on the thread list at level 4 (a reply to a reply to a reply… Whew!). The message list now looks like this:

```
000006-000000
```

The thread list looks like this:

```
000001-000000 At Level 1
000002-000001 At Level 2
000004-000002 At Level 3
000005-000001 At Level 2
000008-000005 At Level 3
000009-000008 At Level 4
```

The routine then goes back one level and looks for more replies to message 8. There are none, so the routine goes back yet another level and looks for replies to message 5. There are none. Again, the routine goes back and looks for replies to message 1. There are no more replies to message 1, and there are no more levels to travel back to, so the routine stops. The thread has been determined.

At this point, a subsequent call to `MakeThreadList` would yield another thread with message 6 all by itself at level 1. If we were to take the threads from all three of these `MakeThreadList` calls and make one big list of messages, this would be the result:

```
000003-000000 At Level 1 (First MakeThreadList)
000007-000003 At Level 2 (First MakeThreadList)
000010-000007 At Level 3 (First MakeThreadList)
000001-000000 At Level 1 (Second MakeThreadList)
000002-000001 At Level 2 (Second MakeThreadList)
000004-000002 At Level 3 (Second MakeThreadList)
000005-000001 At Level 2 (Second MakeThreadList)
000008-000005 At Level 3 (Second MakeThreadList)
000009-000008 At Level 4 (Second MakeThreadList)
000006-000000 At Level 1 (Third MakeThreadList)
```

Notice that the messages within a thread are naturally sorted with the oldest, original message at the top. However, at the top level of display, the threads with the newest posts (the highest message numbers) appear at the top of all the threads. This arrangement is useful because users typically want to see the newest posts first in a list, but they also want to see the relationship of the post as it exists within a thread or topic. If this list were displayed on the BBS, the messages would be indented according to the level they occupy:

```
000003-000000 At Level 1
  000007-000003 At Level 2
    000010-000007 At Level 3
 000001-000000 At Level 1
   000002-000001 At Level 2
     000004-000002 At Level 3
  000005-000001 At Level 2
    000008-000005 At Level 3
      000009-000008 At Level 4
000006-000000 At Level 1
```

As explained previously, the `MakeThreadList` routine starts with an array of all the message filenames in a file list array passed by reference (`@file_list`).

```
sub MakeThreadList
{
local(*file_list) = @_;
```

`@threads` is used as an array to hold the thread that is currently being built. `$seq_ptr` is the number of the element in the file list array that is

currently being looked at. In addition, `$sequence` and `$previous` are used for housekeeping during the processing of the message list.

```
local(@threads,$seq_ptr);
local($sequence,$previous);
```

Initially, the script looks at the highest message number in the file list. If the highest message number exists (the `@file_list` array has elements in it), the routine processes the thread.

```
$seq_ptr = @file_list - 1;
if ($seq_ptr > -1)
{
```

First, the header information from the highest message number filename is read to retrieve the date that the message was posted. When the thread is processed, the `$post_date` is used to display the date of the newest message along with the date of the original post.

```
($poster_firstname, $poster_lastname, $poster_email,
 $post_date, $post_subject, $post_options) =
  &GetMessageHeader("$forum_dir/@file_list[$seq_ptr]");
```

The `while` loop backtracks through all the replies until it reaches the original post in the thread we are examining. When the loop exits, `$sequence` is the message number of the original post and `$seq_ptr` points to the element in the `@file_list` array that corresponds to this message. `GetPointer` is used to constantly get the updated number of the element in the array that the script is pointing to.

```
while(1)
     {
         @file_list[$seq_ptr] .= "|$post_date";
         $sequence = @file_list[$seq_ptr];
         $previous = substr($sequence,7,6);
         $previous_pointer =
          &GetPointer(*file_list, $previous);
     if  (($previous eq "000000") ||
          ($previous_pointer == -1))
          {
```

```
          last;
          }
     $seq_ptr = $previous_pointer;
     } #End of while loop
```

`@seq_stack` is set to the current sequence pointer so that the routine always knows where to begin searching for messages in the `@file_list` array. `$cur_stack_size` is set to 1, which corresponds to the level in the thread that we had discussed earlier. Next, the `@threads` array is initialized and the original message is set to level 1. The level and the message number (`$sequence`) are pipe-delimited inside the `@threads` array.

```
@seq_stack = ($seq_ptr);
$cur_stack_size = 1;
push(@threads, "$cur_stack_size|$sequence");
```

Now that the routine has found the original post of the latest thread in the `@file_list`, the routine goes through the entire `@file_list` array to determine the thread order.

```
while(@file_list > 0)
{
```

`$next_seq` is set to the current message number. This message number is used to find the reply to this message by calling the `GetNextThread` function. `$next_ptr` is equal to the number of the element in the `@file_list` array that corresponds to the first reply to the current message number.

```
$next_seq = substr($sequence,0,6);
$next_ptr =
    &GetNextThread(*file_list, $next_seq, $seq_ptr);
```

If there is a reply (`$next_ptr > -1`), then `$cur_stack_size` is incremented (the level is increased), and this message is placed in the `@threads` array as the next message in the thread.

```
if ($next_ptr > -1)
     {
     $cur_stack_size++;
```

```
     push(@seq_stack, $next_ptr);
     $sequence = $file_list[$next_ptr];
     $seq_ptr = $next_ptr;
     push(@threads, "$cur_stack_size|$sequence");
     }
```

Otherwise, if there is no reply, the last message is removed from the `@file_list` array, and we start looking for replies one level back.

```
else
     {
     @file_list =
         &RemoveElement(*file_list, $seq_ptr);
     $cur_stack_size–;
     pop(@seq_stack);
```

`@seq_stack` is greater than zero if there are more levels of replies to examine.

```
     if (@seq_stack > 0)
          {
          $seq_ptr =
              $seq_stack[@seq_stack - 1];
          $sequence = $file_list[$seq_ptr];
          }
```

If there are no more levels (because we are back to finding more replies to the original post), the `while` loop exits with the `last` command and the thread is complete.

```
     else
          {
          last;
          }
     }
} # End of While Loop
```

Finally, when the `@threads` list is fully generated, it is returned to the procedure that called this function.

```
@threads;

} # End of if seq_ptr > 0
```

If there are no sequence numbers in the array, an empty list is returned for the thread.

```
else {
 ();
} # End of IF Seq_ptr > 0

} # end of MakeThreadList
```

The GetPointer Subroutine

`GetPointer` returns a numerical pointer into an array of files where the sequence number appears as the message number. Remember, message filenames appear as **[message number]=[reply to message number].msg**, where the message number and reply-to number are a fixed six digits.

```
sub GetPointer
{
```

A reference to the `@file_list` array is passed along with the sequence number (`$seq`) to search for.

```
local(*file_list, $seq) = @_;
```

`$x` and `$pointer` are declared local to the subroutine. `$pointer` contains the number of the element in the array that `$seq` corresponds to, and `$x` is used as a temporary variable. Initially, `$pointer` is set to –1. Because arrays count from zero, if `$pointer` is –1, we will that the sequence number has not been found in the file list.

```
local($pointer,$x);
$pointer = -1;
```

The main part of the routine is a `for` loop that iterates through the filenames. If the first six digits of the filename match the sequence number, then `$pointer` is set to `$x` (the current element number) and the `for` loop exits with the `last` command.

```
for ($x = 0;$x < @file_list; $x++)
     {
     if (substr($file_list[$x],0,6) eq $seq)
          {
          $pointer = $x;
          last;
          }
     }
```

Finally, the `$pointer` is returned.

```
$pointer;
} # End of GetPointer
```

The GetNextThead Subroutine

`GetNextThread` retrieves the next reply to a particular message number in the sorted filenames list. The file list is passed by reference as the `@file_list` array. The sequence number of the message (`$seq`) to find replies for is also passed, as is the number of the element in the `@file_list` to start looking for replies (`$start`).

```
sub GetNextThread
{
local(*file_list, $seq, $start) = @_;
```

`$pointer` and `$x` are declared local to the subroutine. `$pointer` will be returned as a pointer to the next reply to the `$seq` message number in the array. `$pointer` is initialized to –1. Remember that arrays start counting at zero, and we want to differentiate between a number that points to the array and one that does not. If `$pointer` was not initialized, it would have a default value of zero which would be a valid pointer into the array. Instead, when `$pointer` is returned as –1, the script knows that there are no more replies to be found for the message number (`$seq`).

```
local($pointer, $x);
$pointer = -1;
```

The array is searched from `$start` to its end using a `for` loop.

```
for ($x = $start; $x < @file_list; $x++)
     {
```

If the second six-digit number in the filename (the reply to message number) is equal to the message number (`$seq`), then `$pointer` is set equal to `$x` as the current element number of the array, and the `for` loop is exited using the `last` command.

```
     if (substr($file_list[$x],7,6) eq $seq)
          {
          $pointer = $x;
          last;
          }
     }
```

The routine ends by returning the `$pointer`.

```
$pointer;

} # End of GetNextThread
```

The RemoveElement Subroutine

The `RemoveElement` subroutine is simple. It takes a reference to an array plus the number of the element to delete from the array and uses Perl's `splice` function to remove the element. Finally, the routine returns the resulting array.

```
sub RemoveElement
{
local(*file_list, $number) = @_;

if ($number > @file_list)
     {
     die "Number was higher than " .
         "number of elements in file list";
     }
splice(@file_list,$number,1);
@file_list;
} # End of RemoveElement
```

The GetForumInfo Subroutine

GetForumInfo takes the forum variable name ($forum) and returns the full descriptive name of the BBS forum as well as the directory where the BBS forum messages are stored.

```
sub GetForumInfo
{
    local($forum) = @_;
```

$forum_name, $forum_dir, $x, and $forum_number are defined as local variables that are used later in the subroutine.

```
    local($forum_name, $forum_dir, $x);
    local($forum_number);
```

Initially, $forum_number is set to –1. At the end of the routine, the script knows that the name was not found in the list of BBS forum names if $forum_number is still –1. If $forum has been found, $forum_number is set to the number of the element in the @forum_variable array in which the name of the forum is defined.

```
$forum_number = -1;
```

The body of the GetForumInfo routine uses a for loop to step through each element in the @forum_variable array.

```
for ($x = 1; $x <= @forum_variable; $x++)
     {
```

If the current element is equal to the contents of $forum, then $forum_number is set to the number of the current element in the array and the for loop exits when it encounters the last command.

```
     if ($forum_variable[$x - 1] eq $forum)
          {
          $forum_number = $x - 1;
          last;
          }
     } # End of FOR forum_variables
```

Now that the array has been processed, $forum_number should no longer be –1. If it is not –1, then $forum_name and $forum_dir are assigned their respective values based on the corresponding elements in the @forums and @forum_directories arrays.

```
if ($forum_number > -1)
{
    $forum_name = @forums[$forum_number];
    $forum_dir = @forum_directories[$forum_number];
```

If $forum_number is still –1, then $forum_name and $forum_dir are cleared. To generate a better error message, $forum is set to "None Given" if $forum is an empty string. $error is set to a message that tells the user that the $forum was not available. Then **bbs_html_error.pl** is loaded to print the error message to the user and the program exits with the die command.

```
} else
{
    $forum_name="";
    $forum_dir = "";
    if ($forum eq "") {
        $forum = "Forum Not Entered";
    }
    $error =
     "<strong>Forum '$forum' Not Found</strong>";
    require "bbs_html_error.pl";
    die;
}
```

If the routine successfully found the BBS forum information, it returns it as an array of two elements: $forum_name and $forum_dir.

```
($forum_name, $forum_dir);

} # end of GetForumInfo
```

The PruneOldMessages Subroutine

The PruneOldMessages subroutine is responsible for removing old messages in a BBS forum directory.

```
sub PruneOldMessages {
```

$forum_dir and a reference to the @files array are sent as parameters to PruneOldMessages. Both variables are declared local to this subroutine. However, the global variables $prune_how_many_days and $prune_how_many_sequences, which are defined in the **bbs.setup**, affect how this routine deletes messages. $x, $prunefile, $attachfile, and $attachfile2 are declared as local variables that will be used at various points during this subroutine.

```
local($forum_dir, *files) = @_;
local($x);
local($prunefile, $attachfile, $attachfile2);
```

The main body of this routine goes through each of the files in the @files array.

```
for ($x = @files; $x >= 1; $x—) {
```

$prunefile is set to the full path and filename of the file that is currently being checked for age.

```
 $prunefile = "$forum_dir/$files[$x - 1]";
```

In addition, $attachfile and $attachfile2 are defined as filenames of attachments that may be associated with the $prunefile message if the user has uploaded a file with the message. $attachfile is the attachment description file in the forum directory, and $attachfile2 is the binary file that was uploaded and is referenced by the attachment description file.

```
$attachfile = "$forum_dir/" .
    substr($files[$x - 1],0,14) .
    "attach";
$attachfile2 = "$attach_dir/" .
    "$forum-" .
    substr($files[$x - 1],0,14) .
    "bin";
```

The -M parameter is used to check the last modification date in days. If it is greater than $prune_how_many_days and if $prune_how_many_days is greater than zero, then the file is deleted and the name is removed from the @files array. In addition, the attachments are removed if they exist.

```
    if ((-M "$prunefile" > $prune_how_many_days) &&
        ($prune_how_many_days > 0)) {
        unlink("$prunefile");
        unlink($attachfile);
           unlink($attachfile2);
        &RemoveElement(*files, $x - 1);
        next;
    }
```

`$x` is the current number of the element that we are processing in the `@files` array. If `$x` is less than or equal to the total number of elements in the array minus the maximum number of sequences to keep around (`$prune_how_many_sequences`) and `$prune_how_many_sequences` is not zero, then the file is deleted and the corresponding element is removed from the `@files` array. Also, the attachments are removed if they exist.

```
    if (($x <= (@files - $prune_how_many_sequences))
        && ($prune_how_many_sequences != 0)) {
        unlink("$prunefile");
        unlink($attachfile);
           unlink($attachfile2);
        &RemoveElement(*files, $x - 1);
        next;
    }
   } # End of for all files
} # End of PruneOldMessages
```

The HiddenFields Subroutine

The `HiddenFields` routine generates a string containing the HTML code for hidden `<INPUT>` tags containing variables that must be passed from screen to screen in WebBBS.

Basically, Web servers see each request to run the WebBBS script as a separate, individual use of WebBBS. The Web server has no idea that you have previously performed actions such as posting a message. To maintain continuity for the user, a series of hidden form fields is placed in each HTML form so that when the user submits an operation to the BBS script, it knows who the user is and can maintain the session the user is in.

`$buf` and `$h` are declared local. `$buf` serves as a continually growing string of hidden input fields. `$h` serves as shorthand for the `"<INPUT`

TYPE=HIDDEN NAME" part of the hidden field tag, because it must be assigned to $buf over and over again in the subroutine.

```
sub HiddenFields {
    local ($buf);
    local ($h);

    $h = "<INPUT TYPE=HIDDEN NAME";
```

The first variable passed is the session ID. It is used by the authentication library to retrieve the user information, such as first and last name, every time the BBS script is called. Chapter 9 discusses the need to pass the session variable from form to form.

```
    $buf = qq!$h=session VALUE="$session">\n!;
    if ($first_date ne "") {
     $buf .=
         qq!$h=first_date VALUE="$first_date">\n!;
    }
```

The next group of hidden variables has to do with filtering of messages. If you will recall, there are certain form variables such as keywords and last_date that let a user choose to filter the list of messages to be displayed. Because the user may interrupt viewing by reading or posting a message, these variables are passed from the Create Message screen back to the list of messages when the forum is printed again.

```
    if ($last_date ne "") {
     $buf .=
         qq!$h=last_date VALUE="$last_date">\n!;
    }
    if ($first_days_old ne "") {
     $buf .=
         qq!$h=first_days_old
          VALUE="$first_days_old">\n!;
    }
    if ($last_days_old ne "") {
     $buf .=
         qq!$h=last_days_old
          VALUE="$last_days_old">\n!;
    }
```

```
    if ($keywords ne "") {
     $buf .=
          qq!$h=keywords VALUE="$keywords">\n!;
    }
    if ($exact_match ne "") {
     $buf .=
          qq!$h=exact_match VALUE="$exact_match">\n!;
    }
```

`$use_last_read` and `$last_read` also result in the filtering of messages and are passed from screen to screen for the same reason. These variables trigger the viewing of only new messages by the user.

```
    if ($use_last_read = "on") {
     $buf .=
          qq!$h=use_last_read
            VALUE="$use_last_read">\n!;
    }
   if ($last_read ne "") {
     $buf .=
          qq!$h=last_read VALUE="$last_read">\n!;
    }
```

If the setup file being used is an alternative one, the setup information must always be passed from screen to screen in the BBS. The `setup` form variable must be defined so that the alternative setup information can be read by each subsequent call to the BBS script.

```
    if ($setup_file ne "") {
     $buf .=
          qq!$h=setup VALUE="$setup_file">\n!;
    }
```

Finally, `$buf` is returned with all the appropriate hidden fields.

```
    $buf;
} # End of Hidden Fields
```

The ReadMessageFields Subroutine

`ReadMessageFields` sets up a list of URL variables that need to be passed from screen to screen in the BBS script. Basically, to read a message, the

user must click on a URL link to that message. However, when you click on a URL link instead of doing a form submission, not all the normal <INPUT> tag form variables are passed. Thus, the URL for each message must include certain common form variables. These are the same basic variables that are generated as hidden <INPUT> tags in the HiddenFields routine.

$buf is declared local. $buf contains the extra variables that belong in the URL.

```
sub ReadMessageFields {
    local ($buf);
```

As in the HiddenFields subroutine, here $buf is set to a number of variables. In this case, the variables need to show up on a URL line and are generated as variable=value pairs instead of as hidden <INPUT> tags.

```
    $buf = qq!session=$session&!;
    if ($first_date ne "") {
     $buf .=
         qq!first_date="$first_date"&!;
    }
    if ($last_date ne "") {
     $buf .=
         qq!last_date=$last_date&!;
    }
    if ($first_days_old ne "") {
     $buf .=
         qq!first_days_old=$first_days_old&!;
    }
    if ($last_days_old ne "") {
     $buf .=
         qq!$last_days_old=$last_days_old&!;
    }
    if ($keywords ne "") {
     $buf .=
         qq!keywords=$keywords&!;
    }
    if ($exact_match ne "") {
     $buf .=
         qq!exact_match=$exact_match&!;
    }
    if ($use_last_read = "on") {
     $buf .=
         qq!use_last_read=$use_last_read&!;
    }
```

```
    if ($last_read ne "") {
     $buf .=
          qq!last_read=$last_read&!;
    }
    if ($setup_file ne "") {
     $buf .=
          qq!setup=$setup_file&!;
    }
```

Because `$buf` will be placed on a URL, certain variables need to be filtered from the URL. Spaces and forward slash (/) characters are among the most common ones encountered, so we filter them. Spaces are replaced with `%20`, and forward slashes are replaced with `%2F`.

```
    $buf =~ s/ /%20/;
    $buf =~ s/\//%2F/;
```

Finally, because all the fields were appended with an ampersand (`&`) character, the very last ampersand is `chop`ped off and the resulting buffer is returned.

```
    chop($buf);
    $buf;
} # End of ReadMessageFields
```

Bbs_html_error.pl

Bbs_html_error.pl is a Perl library that contains code to print an HTML screen to the user's Web browser informing the user that an error has occurred. Figure 25.4 shows an example of the output from **Bbs_html_error.pl**.

The error message is defined in `$error`, which is printed between the HTML `<Blockquote>` tags.

```
print <<__ENDOFERROR__;
<HTML>
<HEAD>
<TITLE>Problem In BBS Occurred</TITLE>
</HEAD>
```

```
<BODY>
<h1>Problem In BBS Occurred</h1>
<HR>
<blockquote>
$error
</blockquote>
<HR>
</BODY>
</HTML>
__ENDOFERROR__
```

Bbs_html_forum.pl

Bbs_html_forum.pl is a library that prints the HTML screen for the list of messages in a given forum. The program relies on several variables having been defined previously: `$message_html`, `$forum_name`, `$print_hidden_fields`, `$create_message_error`, `$bbs_buttons`, and `$bbs_script`. The output of this routine can be seen in Figure 25.1.

The HTML related to all the individual messages is contained in the `$message_html` variable. `$forum_name` is the descriptive name of the current forum. `$print_hidden_fields` are set up as hidden input form fields that pass certain variables, such as session ID, from screen to screen in the BBS script. The forum name is treated as a separate hidden field, because the `$print_hidden_fields` string may be used in situations in which the forum name is not appropriate. For example, a screen might be added by a programmer who wants to allow the user to change forums dynamically without going back to the front page. `$create_message_error` is a message that tells the user that a previous post was unsuccessful. `$bbs_buttons` is set up in the **bbs.setup** file as the URL directory that contains the images for the BBS buttons. `$bbs_script`, also defined in **bbs.setup**, is the filename of the main BBS script (**bbs_forum.cgi**).

```
$print_hidden_fields = &HiddenFields;
print <<__END_OF_HTML__;
<HTML><HEAD>
<TITLE>The Message Board</TITLE>
</HEAD>
<BODY BGCOLOR = "FFFFFF" TEXT = "000000">
<CENTER>
```

```
<H2>Welcome To The $forum_name Message Board</H2>
</CENTER>
$create_msg_error
<HR>
<B>Messages:</B>
<BR>
$message_html
<CENTER>
<P>
<FORM ACTION=$bbs_script METHOD=POST>
<INPUT TYPE=HIDDEN NAME=forum VALUE=$forum>
$print_hidden_fields
<HR>
<P>
<INPUT TYPE=IMAGE NAME=post_op SRC="$bbs_buttons/post.gif" BORDER=0>
</CENTER>
</FORM>
</BODY>
</HTML>
__END_OF_HTML__
```

Bbs_html_read_message.pl

Bbs_html_read_messsage.pl is a Perl library that prints the HTML code for reading an individual message on the BBS. The main BBS CGI script defines many variables before calling this routine.

The `$poster_firstname`, `$poster_lastname`, `$poster_email`, `$post_date-time`, `$post_subject`, and `$post_message` variables are filled with the various header fields and body of the main message. `$post_replies` is a list of URLs that correspond to the replies to this message. `$post_attach_html` is a list of URLs that correspond to any attachments that the user may have uploaded. Figure 25.5 shows an example of the output from **Bbs_html_read_message.pl**.

As in **Bbs_html_forum.pl**, `$print_hidden_fields` is used to pass certain variables, such as session id, from BBS screen to screen. Additional hidden fields that are not universal to all WebBBS screens are also included in the following HTML code. `$bbs_buttons` is the URL directory containing the images for the BBS buttons. `$bbs_script` is the main CGI script name (**bbs_forum.cgi**). `$forum` is the current forum name. And finally, `$message` is the current message number. This information is passed to the

BBS script when this HTML form is submitted in order to keep track of which message the user is replying to if the user presses the reply button.

```
$print_hidden_fields = &HiddenFields;

print <<__ENDOFHTML__;
<HTML>
<HEAD>
<TITLE>Reading Messages</TITLE>
</HEAD>
<BODY BGCOLOR = "FFFFFF" TEXT = "000000">

<TABLE BORDER=0>
<TR>
<TH align=left>Name:</TH>
<TD>$poster_firstname $poster_lastname</TD>
</TR>
<TR>
<TH align=left>E-Mail:</TH>
<TD><A HREF="mailto:$poster_email">$poster_email</A></TD>
</TR>
<TR>
<TH align=left>Date/Time:</TH>
<TD>$post_date_time</TD>
</TR>
<TR>
<TH align=left>Subject:</TH>
<TD>$post_subject</TD>
</TR>
</TABLE>
<B>Body:</B>
<BLOCKQUOTE>
$post_message
</BLOCKQUOTE>
<FORM METHOD=POST ACTION="$bbs_script">
<B>Replies:</B>
$print_hidden_fields
<INPUT TYPE=HIDDEN NAME=post_subject VALUE="$post_subject">
<INPUT TYPE=HIDDEN NAME=forum VALUE="$forum">
<INPUT TYPE=HIDDEN NAME=reply_to_message VALUE="$message">
<BLOCKQUOTE>
$post_replies
</BLOCKQUOTE>
$post_attach_html
<CENTER><P>
<INPUT TYPE=IMAGE NAME=reply_op SRC="$bbs_buttons/post_a_reply.gif"
```

```
BORDER="0">
<INPUT TYPE=IMAGE NAME=post_op SRC="$bbs_buttons/post.gif"
BORDER="0">
<INPUT TYPE=IMAGE NAME=toplevel_op
SRC="$bbs_buttons/message_archive_top.gif"
BORDER="0">
</CENTER>
</FORM>
</BODY>
</HTML>
__ENDOFHTML__
```

Bbs_html_create_message.pl

Bbs_html_create_message.pl is called by **bbs_forum.cgi** to print an HTML form for creating a message. The Create Messsage HTML form serves the needs of both posting and replying to posts. The only difference is that the reply has additional fields that carry information from the post being replied to. An example of the output from **Bbs_html_create_message.pl** is shown in Figures 25.6 (an original post) and 25.12 (a reply to a post).

Most of the input fields are blank by default. However, depending on the logic that the main BBS script encounters, it may place default values into the various `<INPUT>` fields in this HTML form. `$post_first_name_field`, `$post_last_name_field`, `$post_email_field`, `$post_date_time`, `$post_subject`, and `$post_message` are all possible fillers for the corresponding fields on the HTML form. For example, if this message is a reply to another one, the contents of `$post_message` will be filled with the previous message so that users can quote the person they are replying to.

If attachments are turned on, `$post_attachment` will contain the text for the HTML of the new `<INPUT TYPE=FILE>` HTML tag. If `$post_attachment` has this value, the form header (`$form_hdr`) is assigned a string that contains the multipart encoding type tag in it. If, however, `$post_attachment` is empty, the default encoding type for the form is used. If you recall from the chapter on **Cgi-lib.pl**, an encoding type of `multipart` is necessary for processing file uploads.

`$post_want_email` contains the HTML code for an `<INPUT>` check box where users can choose to receive e-mailed replies automatically. This variable will be blank if you have configured the BBS not to allow the user to choose E-mail replies.

`$title` is set to the title of the Create Message screen. `$forum` is the forum name. `$reply_to_email` is the user's E-mail address; if the post generates automatic E-mail replies, this address will be stored along with the message when it is posted.

The default hidden fields are included with the `$print_hidden_fields` string. Additionally, the hidden fields related to forum and reply information are present in the HTML code.

As usual, after all the variables have been set up, the HTML code is printed using the `HERE DOCUMENT` method.

```
$print_hidden_fields = &HiddenFields;

$form_hdr =
  qq!<FORM METHOD=POST ACTION="$bbs_script" !;
$form_hdr .= qq!ENCTYPE="multipart/form-data">!;

if ($post_attachment eq "") {
$form_hdr = qq!<FORM METHOD=POST
              ACTION="$bbs_script">!;
}
print <<__ENDHTML__;
<HTML>
<HEAD>
<TITLE>$title</TITLE>
</HEAD>
<BODY BGCOLOR = "FFFFFF" TEXT = "000000">
<H2>$header</H2>
<HR>
$form_hdr
<INPUT TYPE=HIDDEN NAME=reply_to_message VALUE="$reply_to_message">
<INPUT TYPE=HIDDEN NAME=forum VALUE="$forum">
$print_hidden_fields
<INPUT TYPE=HIDDEN NAME=reply_to_email VALUE="$reply_to_email">
<TABLE>
<TR>
<TH align=right>First Name:</TH>
<TD>$post_first_name_field</TD>
```

```
</TR><TR>
<TH align=right>Last Name:</TH>
<TD>$post_last_name_field</TD>
</TR>
<TR>
<TH align=right>Email:</TH>
<TD>$post_email_field</TD>
</TR>
<TR>
<TH align=right>Date:</TH>
<TD>$post_date_time</TD>
</TR>
<TR>
<TH align=right>Subject:</TH>
<TD><INPUT TYPE=text NAME=form_subject VALUE="$post_subject"
SIZE=40 MAXLENGTH=50></TD>
</TR>
</TABLE>
<STRONG>Enter Message Below:</STRONG><BR>
<TEXTAREA NAME=form_message ROWS=10 COLS=60
WRAP=physical>
$post_message
</TEXTAREA>
$post_attachment
$post_want_email
<CENTER>
<HR>
<INPUT TYPE=SUBMIT NAME=create_message_op
VALUE="Submit Message">
<INPUT TYPE=RESET VALUE="Clear Values">
<HR>
<P>
<INPUT TYPE=IMAGE NAME=toplevel_op
SRC="$bbs_buttons/message_archive_top.gif"
BORDER="0">
<P>
</CENTER>
</FORM>
</BODY>
</HTML>
__ENDHTML__
```